SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Nick Findrik

Zagreb, 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Prof. dr. sc. Darko Kozarac, dipl. ing.

Student:

Nick Findrik

Zagreb, 2018.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se svom mentoru prof. dr. sc. Darku Kozarcu na stručnoj pomoći, uputama i savjetima tokom izrade završnog rada. Također se zahvaljujem Marcelu Besediću i Mateu Milovcu na velikoj pomoći u izradi shema i Marcelu na projektiranju tiskane pločice.

Nick Findrik

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
procesno-energetski, konstrukcijski, brodostrojarski i inženjersko modeliranje i računalne simulacije

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur. broj:	

DIPLOMSKI ZADATAK

Student: **Nick Findrik** Mat. br.: 0035190239

Naslov rada na hrvatskom jeziku: **Izrada računala motora bolida Formule student**

Naslov rada na engleskom jeziku: **Development of Engine Control Unit of Formula Student Car**

Opis zadatka:

Rad suvremenih motora s unutarnjim izgaranjem u velikoj mjeri ovisi o takozvanom računalu motora (ECU – engine control unit). Glavni zadaci računala motora su upravljanje ubrizgavanjem goriva te upravljanje preskakanjem iskre na svjećici. Pored toga računalo ima zadatak da nadzire rad motora pomoću informacija koje dobiva od raznih senzora, npr. temperatura rashladne tekućine.
Motor koji se koristi na bolidu Formule student značajno je izmijenjen u odnosu na izvornu verziju. Stoga je potrebno napraviti potpuno novo upravljanje, tj. napraviti novo računalo motora. Jedna mogućnost je da se na motor postavi jedan od komercijalno dostupnih programibilnih računala kao što su: MoTeC, VEMS, itd. Druga mogućnost je da se izradi vlastito računalo motora koje će upravljati svim definiranim zadaćama. U okviru ovog zadataka potrebno je izraditi vlastito računalo motora Formule student. U okviru rada potrebno je:

- Napraviti pregled komercijalno dostupnih programabilnih računala motora te definirati prednosti i nedostatke pojedinih.
- Napraviti popis svih zadataka koje računalo motora Formule student mora odraditi.
- Napraviti pregled svih senzora koji će se spajati na računalo motora te prikazati vrste signala koji se s istih dobivaju.
- Napraviti pregled svih aktuatora kojima računalo motora mora upravljati te prikazati vrste signala kojima se s istim aktuatorima upravlja.
- Izraditi shemu računala motora.
- Izraditi upravljački program pomoću kojeg će definirani hardware upravljati radom motora.
- Testirati sustav i prikazati da isti radi onako kako je definirano projektnim zadatkom.

Pri izradi se treba pridržavati uobičajenih pravila za izradu diplomskoga rada.
U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datum obrane:
18. siječnja 2018.	22. ožujka 2018.	28., 29. i 30. ožujka 2018.

Zadatak zadao: Predsjednica Povjerenstva:

Prof. dr. sc. Darko Kozarac Prof. dr. sc. Tanja Jurčević Lulić

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS TEHNIČKE DOKUMENTACIJE

ECU-0-0	Glavni list
ECU-1-0	Analogni ulazi
ECU-1-1	Analogni ulaz GP
ECU-1-2	WBO2 kontroler
ECU-2-0	Komunikacija
ECU-2-1	UART-USB
ECU-3-0	Konektori
ECU-4-0	Digitalni ulazi
ECU-4-1	Digitalni ulazi GP
ECU-4-2	VR conditioner
ECU-5-0	Izlazni kanali
ECU-5-1	Relay driver
ECU-5-2	Driver brizgaljki i bobina
ECU-5-3	HP izlazni kanali
ECU-6-0	EEPROM
ECU-7-0	Mikrokontroler
ECU-8-0	Regulatori napona
ECU-9-0	Zaštita vanjskih 5V

POPIS OZNAKA

Oznaka	Jedinica	Opis
ADC	-	Očitana vrijednost analogno-digitalnog pretvarača
C	F	Kapacitet kondenzatora
dwell	μs	Duljina punjenja bobine
f	Hz	Frekvencija
fi	°	Kut pretpaljenja
I	A	Jakost struje
Ip	mA	Struja transportnog članka lambda sonde
k	$W/(m^2K)$	Koeficijent prijelaza topline
L	H	Induktivitet zavojnice
m_g	kg	masa goriva
m_z	kg	masa zraka
p	Pa	tlak
PW	ms	duljina otvorenosti brizgaljke
R	Ω	Električni otpor
R	J/molK	individualna plinska konstanta
RPMsplit	μs	Udaljenost 2 zuba na enkoderu
T	K	Temperatura
V_{cyl}	m^3	volumen cilindra
VE	-	volumetrijska efikasnost
V_{RS}	V	Napon induktivnog senzora
Z	-	maseni omjer zraka i goriva
λ	-	Faktor pretička zraka
λ	W/(mK)	Toplinska vodljivost
v	m^2/s	Kinematička viskoznost
τ	s	Vremenska konstanta
Φ	W	Toplinski tok

SAŽETAK

U sklopu ovog rada izrađena je centralna jedinica motora bolida formule student. Proučena su postojeća komercijalno dostupna programibilna računala motora, definirani njihovi prednosti i nedostatci te je napravljen popis zadataka i funkcija koje računalo mora odraditi na bolidu formule student. Odlučeno je da će se napraviti računalo s mogućnosti upravljanja motorom do 4 cilindra iako se na trenutnom bolidu koristi jednocilindrični motor. Sama kompleksnost sustava ne povećava se značajno, ali se omogućuje testiranje na četverocilindričnom motoru prije završetka izrade bolida. Napravljena je analiza svih senzora i aktuatora koji se koriste na bolidu i definirani su strujni krugovi i komponente potrebne za kondicioniranje signala i upravljanje aktuatorima. Za sve komponente korištena su SMD (komponente za površinsku montažu) pakiranja koja zbog manjih dimenzija omogućuju veću gustoću komponenti na tiskanim pločicama, a time i manje dimenzije tiskane pločice. Druga velika prednost je znatno veća otpornost na vibracije, posebno bitno kod jednocilindarski motora trenutno korištenog na bolidu. Sheme su napravljene u programskom paketu Altium Designer, a prateće komponente integriranih krugova odabrane prema preporukama proizvođača ili proračunu. Prije izrade konačne verzije, glavne komponente ispitane su zasebno i izrađena je centralna jedinica na prototipnoj pločici u svrhu testiranja upravljačkog programa mikrokontrolera. Upravljački program za mikrokontroler STM32F4 napisan je C++ programskim jezikom u Visual Micru uz korištenje stm32duino biblioteka. Kućište centralne jedinice izrađeno je aditivnim postupkom selektivnog laserskog sinteriranja polimera, a spoj dva dijela kućišta brtvljen je gumenom brtvom kako bi se osigurala vodonepropusnost. Na kraju je u Visual Basicu napravljena aplikacija koja preko USB porta komunicira s centralnom jedinicom i omogućuje jednostavno podešavanje parametara ubrizgavanja, paljenja i raznih ostalih funkcija u realnom vremenu te nadzor i prikupljanje svih podataka na računalo. Prototipna verzija sustava testirana je na četverocilindričnom motoru iz Honde Civic 1997.

Ključne riječi: ubrizgavanje goriva, upravljačka jedinica motora, paljenje smjese, tiskana pločica, motor s unutarnjim izgaranjem, formula student

SUMMARY

In this thesis the engine control unit of the Formula Student car was developed. Commercially available programable engine control units have been studied, their advantages and disadvantages were defined and a list of tasks and functions that an ECU must perform on a formula student car has been made. It was decided that a ECU with a 4-cylinder engine management capability will be designed even though a single cylinder engine is used on the current formula student car. In this way the system's complexity does not increase significantly, but it is possible to test the ECU on a readily available four-cylinder engine before the formula student car is finished. An analysis of all sensors and actuators used on the car was performed and the circuitry and components required for signal conditioning and actuator control were defined. SMD (surface mount device) components were used for all circuits which due to smaller dimensions, allow for greater component density and thus smaller dimensions of the printed circuit board. Another advantage is significantly higher resistance to mechanical vibrations, especially important for the one-cylinder engine currently used on the car. Electrical schematics were made in the Altium Designer software package and the accompanying components of the integrated circuits were selected according to manufacturer's recommendation or calculation. Before the final version was made, the main components were tested separately and an engine control unit was built on the prototyping board to test the microcontroller firmware. The STM32F4 microcontroller firmware was written in C++ programming language in Visual Micro using the stm32duino library. The ECU case was made by additive manufacturing selective laser sintering of the polymer, and the connection of the two housing parts was sealed with a rubber seal to ensure waterproofness. Finally, in Visual Basic, an application that communicates with the engine control unit via the USB port was made, which allows simple adjustments of injection, ignition and various other parameters in real-time and monitoring and logging of all the data to the PC. The prototype version of the system was tested on a four-cylinder engine from Honda Civic 1997.

Key words: fuel injection, engine control unit, ignition, printed circuit board, internal combustion engine, formula student

1. UVOD

Formula Student međunarodno je studentsko natjecanje u razvoju, konstrukciji, proizvodnji i prezentaciji bolida izrađenog prema pravilima definiranim Formula SAE pravilnikom. Pravilnikom su propisana dozvoljena konstrukcijska rješenja na bolidu te potrebne sigurnosne mjere kako bi se opasnost za vozača i ostale sudionike na natjecanju svela na minimum.

Jedno od ograničenja čija je svrha ograničavanje snage motora jest restriktor na usisnoj cijevi. Pravilnikom je određen najveći dopušteni promjer restriktora od 20mm za motore koji za gorivo koriste benzin te 19mm za motore koji koriste mješavinu etanola i benzina(E85). Uz to jedan od elemenata koji se boduje na natjecanju je i efikasnost tj. potrošnja goriva zbog čega je veoma bitno optimirati rad motora u svim režimima rada kako bi se postigla optimalna kombinacija snage i potrošnje.

Motor koji pogoni najnoviji bolid Strix je Husqvarna TE610, radnog volumena 570cm^3 originalno korišten na motociklu istog imena. U tvorničkoj izvedbi motor je bio opremljen rasplinjačem i tranzistorskim sustavom paljenja, ali je zbog potrebe veće fleksibilnosti i podesivosti prepravljen na sustav s ubrizgavanjem goriva kontroliran programibilnom centralnom jedinicom motora(*engine control unit*, ECU). Trenutno korišteni ECU je Vems koji pomoću podataka iz raznih senzora upravlja ubrizgavanjem goriva, paljenjem i ostalim sporednim elementima. Zbog problema s nepouzdanosti Vemsa, slabih mogućnosti prikupljanja podataka u vožnji i nedostatka kvalitetnih uputstava i podrške za korištenje *software*-a odlučeno je da se za sljedeću verziju bolida Strix R nabavi kvalitetniji ECU ili razvije vlastiti. S obzirom na relativno visoku cijenu ECU-a koje se kreću od 1000$ na više te eventualni plus koji bi to moglo donijeti na statičkom dijelu natjecanja gdje se ocjenjuju konstrukcijska rješenja, odlučeno je da će se razviti vlastiti.

U skladu s time cilj ovog rada je napraviti ECU koji može upravljati ubrizgavanjem goriva, preskakanjem iskre i ostalim pomoćnim elementima, ima mogućnost podešavanja svih potrebnih parametara u realnom vremenu, prikupljanja podataka za kasniju analizu te ulaze za sve potrebne senzore i odgovarajuće zaštite na ulazima u slučaju nepravilnog spajanja. Kućište i konektori moraju biti vodootporni,a tiskana pločica otporna na vibracije. Dodatna prednost izrade vlastitog ECU-a je mogućnost jednostavnog dijagnosticiranja i otklanjanja kvara što nije slučaj s kupovnom centralnom jedinicom.

2. ECU općenito

Glavna funkcija ECU-a je upravljanje količinom ubrizganog goriva i trenutkom preskakanja iskre, a na modernim motorima i čitavim nizom drugih komponenata i aktuatora. Zadatci ECU-a mogu se podijeliti na tri osnovna dijela:

1. prikupljanje podataka sa senzora koji fizikalne veličine pretvaraju u električni signal
2. obrada podataka na temelju algoritama u otvorenoj ili zatvorenoj petlji. Upravljački program koji sadrži algoritme spremljen je u memoriji mikrokontrolera koji je mozak cijelog sustava
3. upravljanje aktuatorima pomoću izlaznih električnih signala

Od cijelog sustava zahtijeva se da besprijekorno funkcionira u širokom rasponu temperatura (-40 do +125°), pri jakim vibracijama, padovima napona kod hladnog pokretanja i naglim skokovima napona u slučaju otpajanja akumulatora.

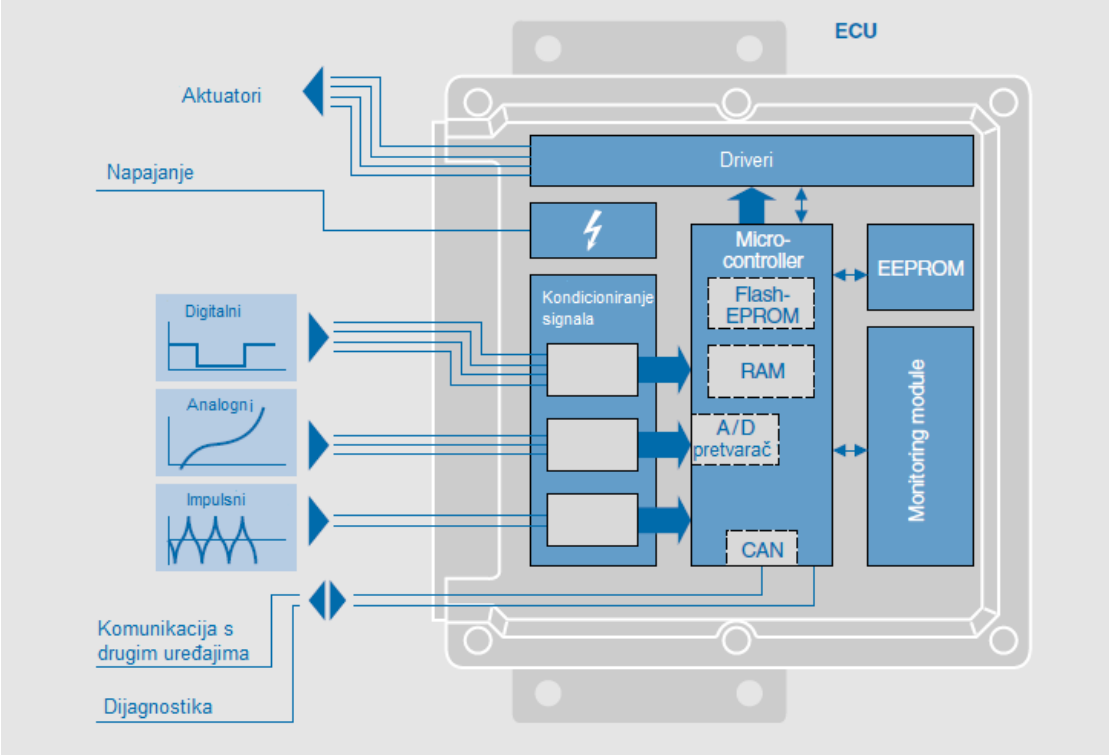
2.1. Ulazni signali

Ulazni električni signali sa senzora mogu se podijeliti u dvije kategorije: analogni i digitalni. Analogni signali mogu imati praktički bilo koju vrijednost napona u određenom rasponu, a primjeri senzora sa analognim signalima su: senzor masenog protoka zraka (MAF senzor), senzor tlaka u usisnom kolektoru (MAP senzor), senzor kuta zaklopke, razni senzori temperature i položaja, napon akumulatora itd. Analogni signali prije ulaza u mikrokontroler prolaze razne zaštite kako bi se spriječilo oštećenje mikrokontrolera zbog slučajnih skokova napona iznad dozvoljene razine, filtre za uklanjanje smetnji i u nekim slučajevima pojačala ako su signali iz senzora preslabi za izravno očitavanje. Analogni signali prije očitavanja prolaze kroz ADC (*analog-digital converter*) koji pretvara signale u digitalni oblik razumljiv centralnoj jedinici mikrokontrolera.

Digitalni signali mogu poprimiti samo dvije vrijednosti, logički 1 ili logički 0. Primjeri senzora sa digitalnim izlaznim signalima su razni prekidači i Hallovi senzori. Digitalne signale za razliku od analognih centralna jedinica mikrokontrolera može izravno obrađivati.

2.2. Mikrokontroler

Mikrokontroler je centralna komponenta svakog ECU-a, a sadrži centralnu procesorsku jedinicu (CPU), ulazne i izlazne kanale, brojače, RAM, ROM, kanale za komunikaciju i ostale popratne sklopove koji se nalaze u jednom integriranom krugu. Da bi mikrokontroler izvršavao svoju zadaću potreban mu je program pohranjen u ROM-u (*read-only memory*) kojeg CPU interpretira kao naredbe i izvršava jednu iza druge. Program sadrži sve algoritme i podatke potrebne za upravljanje radom motora na temelju podataka dobivenih iz senzora i nije ga moguće mijenjati tijekom rada. Tijekom izvršavanja programa varijable se učitavaju u RAM koji zadržava memoriju dokle god ima napajanje. Varijable u RAM-u mogu se mijenjati u realnom vremenu, ali se brišu iz memorije kod prekida napajanja. Ako ih se želi trajno zadržati potrebno ih je pohraniti u EEPROM i zatim na svakom pokretanju učitati u RAM.



Slika 1. Općeniti blok dijagram centralne jedinice motora [4]

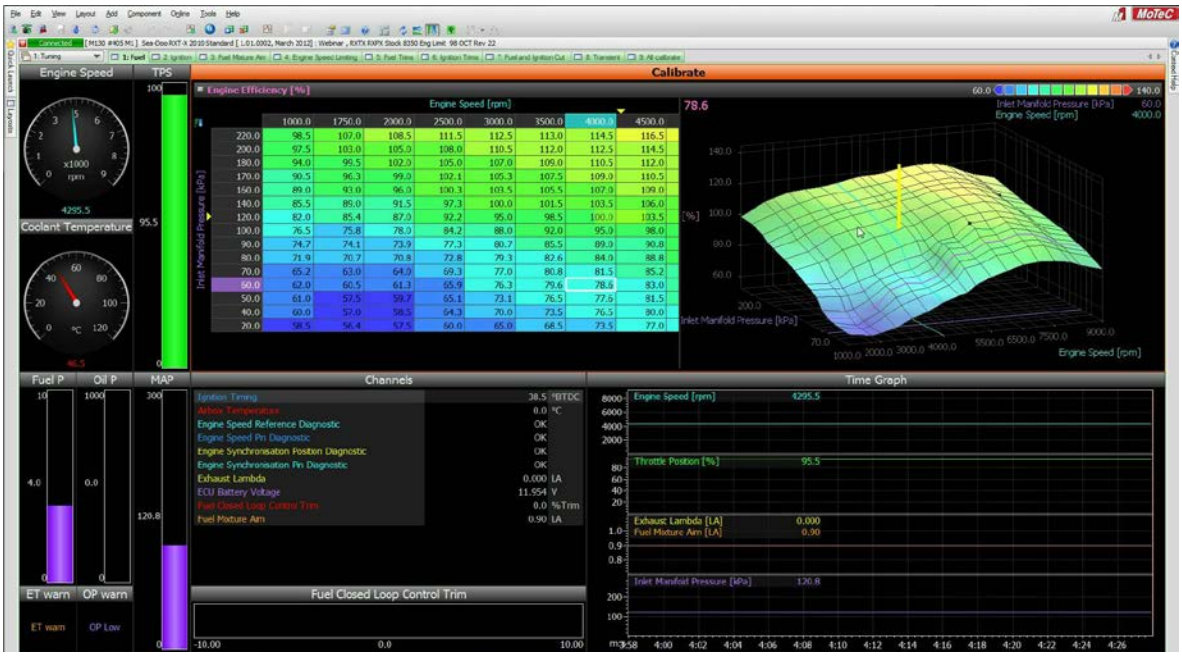
2.3. Izlazni signali

Izlazni signali na mikrokontroleru su napona 3.3v ili 5v uz najveće dopuštene struje do 20mA po izlazu zbog čega je njime nemoguće izravno upravljati aktuatorima. Zbog toga se koriste tranzistori (unipolarni ili bipolarni) pomoću kojih mikrokontroler s relativno malim strujama

može prekidati znatno veće struje. Poželjno je da svi izlazi imaju zaštite od kratkog spoja i detekciju otvorenog kruga u slučaju kvara ili otpajanja nekog od akturatora.

2.4. Komercijalne programibilne centralne jedinice motora

Centralne jedinice motora serijskih vozila tvornički su programirane za određene konfiguracije motora i nisu namijenjene za naknadno podešavanje. Kod bilo kakvih većih modifikacija motora koje mijenjaju stupanj punjenja cilindra, a time i zahtjeve za količinom ubrizganog goriva i kuta pretpaljenja postoji potreba za programabilnom centralnom jedinicom. Njima se omogućuje jednostavno podešavanje i praćenje svih potrebnih parametara motora u realnom vremenu. To uključuje upravljanje trajanjem otvorenosti brizgaljke i trenutkom preskakanja iskre prema glavnim mapama i nizom korekcijskih faktora definiranim od strane korisnika. Da bi upravljanje različitim motorima bilo moguće potrebna je kompatibilnost s raznim tipovima senzora i aktuatora te mogućnost njihove kalibracije. Većina programabilnih ECU-a također posjeduje funkcije specifične za motorsport kao što su *launch control*, *anti-lag* i *ignition shift cut*.



Slika 2. Grafičko sučelje MoTeC

Na tržištu postoji veliki broj proizvođača ovakvih centralnih jedinica, a najčešće svaki proizvođač ima nekoliko različitih modela ovisno o kompleksnosti motora kojim mogu upravljati. Bolid Strix pogonjen je relativno jednostavnim jednocilindarskim motorom zbog čega ga je moguće upravljati gotovo bilo kojim ECU-om, a potrebne funkcije su: upravljanje

ubrizgavanjem goriva i preskakanjem iskre, paljenje pumpe goriva i ventilatora za hlađenje, *launch control* i *ignition shift-cut*. U sljedećoj tablici navedeni su neki od komercijalno dostupnih ECU-a i njihove glavne specifikacije.

Tablica 1. Usporedba programabilnih centralnih jedinica motora

ECU	MoTeC M400	DTA S60Pro	PE3-SP000	EFI Euro4	VEMS
Cijena	2500-4000€	1000€	700€	1800€	500€
Broj cilindara(sekvencionalno)	4	4	2	4	8
Izlazi kanali					
Brizgaljke	4	4	2	8	8
Bobine	4	4	2	4+4	8
Izlazi opće namjene	8	6	10	8	
3D mape za svaki cilindar	Da	Ne	Ne	?	Ne
Ulazni kanali					
Ulazi senzora KV i bregaste osovine	2 (induktivni & Hall)	2+2 (induktivni & Hall)	2(mag & hall)	4+4 (induktivni & Hall)	2
Analogni ulazi	12	12	6	15	9
Digitalni ulazi	4	4	4	7	4
WBO2 kontroler	Opcija	Vanjski kontroler	Vanjski kontroler	Da	Da
Komunikacija					
RS232	Da	Da	Da		Da
CAN	Da	Da	Da	Da	Ne
Prikupljanje podataka					
Data logging	Da	Da	Da	Da	Da
Frekvencija	200Hz	100Hz	100Hz	200Hz	20Hz
Funkcije					
Drive-by-wire	Opcija	Ne	Ne	Da	Da
Upravljanje varijabilnim razvodom	Opcija	Da	Ne?	Da	Da
Traction & launch control	Opcija	Da	Da	Da	Da
Boost control	Da	Da	Da	Da	Da
Kontrola detonacije	Da	Ne	Ne	Da	Da
Shift ignition cut	Opcija	Da	Da	Da	Da
4D mape ubrizgavanja i paljenja	Da	Ne	Ne	?	Ne

Iz tablice je vidljivo da svaki od pet navedenih ECU-a zadovoljava prethodno navedene funkcije. Glavne razlike su u mogućnostima softwarea, analizi prikupljenih podataka i korisničkoj podršci gdje se može reći da MoTeC opravdava ovako visoku cijenu. Drugi razlog relativno velike razlike u cijeni je što su MoTeC i Euro EFI provjereni sustavi korišteni u profesionalnim oblicima motorsporta.

3. Hardware, komunikacija sa senzorima i aktuatorima

Prije nego što se krene u odabir komponenata i izradu shema potrebno je proučiti signale svih senzora koji se koriste na motoru te akturatore kojima će se upravljati. Zatim se mogu definirati strujni krugovi za kondicioniranje ulaznih signala i *driveri* za upravljanje aktuatorima. U ovom dijelu opisat će se svi senzori i aktuatori korišteni na motoru bolida Strix i oni koji bi se mogli koristiti u budućnosti te pripadajući strujni krugovi potrebni za povezivanje svih elemenata s mikrokontrolerom. ECU možemo podijeliti na četiri glavne cjeline:

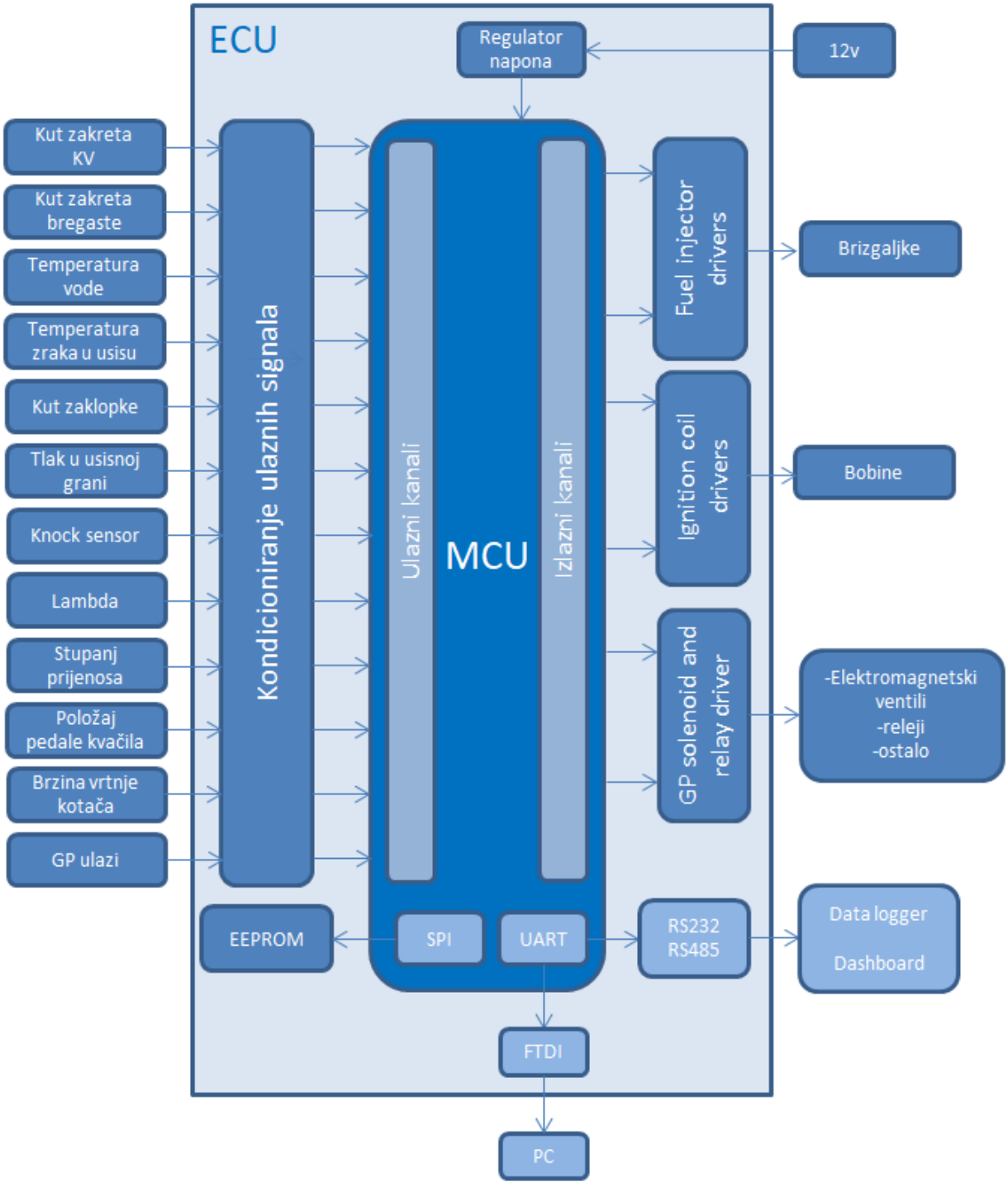
1. Napajanje/regulator napona sa zaštitama od prenapona i obrnutog polariteta
2. Strujni krugovi za kondicioniranje ulaznih signala
3. Driveri za brizgaljke, bobine, releje i ostale komponente
4. Mikrokontroler

Regulator napona služi za napajanje mikrokontrolera i ostalih integriranih krugova koji najčešće rade na naponima od 3.3V ili 5V. Napon sa akumulatora potrebno je spustiti na odgovarajuću razinu i zaštititi sve komponente od mogućih skokova napona i obrnutog polariteta. Isti regulator koristi se za napajanje svih senzora koji rade na 5V pa je bitno osigurati dovoljno veliku raspoloživu struju za sve priključene komponente.

Kondicioniranje ulaznih signala potrebno je zato što mikrokontroler na analognim ulazima može čitati napone od 0-3.3V ili 0-5V, što znači da svi senzori koji promjenom mjerenih veličina mijenjaju otpor ili struju zahtijevaju dodatne strujne krugove koji će te signale pretvoriti u napon. Primjeri senzora što na izlazu daju napon te ih je moguće izravno mjeriti su senzori tlaka u usisnom kolektoru, senzor masenog protoka zraka i senzor položaja zaklopke. S druge strane senzori temperature su otpornici koji promjenom temperature mijenjaju otpor. Oni zahtijevaju dodatan otpornik u ECU-u s kojim tvore djelitelj napona i na izlazu daju napon. Postoji još niz drugih senzora koji će detaljno biti opisani kasnije.

Brizgaljke, bobine, razni elektromagnetski ventili i ostale slične komponente rade na naponu od 12V i sa znatno većim strujama od onih koje može dati mikrokontroler. Za njihovo aktuiranje zato se koriste tranzistori, najčešće bipolarni tranzistori, MOSFET-i ili IGBT-ovi, koji sa miliamperskim strujama na ulazu mogu prekidati nekoliko redova veličina veće struje u izlaznom krugu. Veći potrošači koji ne zahtijevaju brzu aktuaciju moguću s tranzistorima

upravljaju se relejima aktiviranim pomoću tranzistora kako bi se izbjeglo dovođenje velikih struja unutar ECU-a koje mogu izazvati smetnje na signalima sa senzora . Slika 3. prikazuje blok shemu ECU-a sa senzorima i aktuatorima koji će se koristiti na bolidu Strix R. U nastavku će biti objašnjen njihov rad kao i strujni krugovi potrebni za kondicioniranje ulaznih signala i upravljanje akturatorima.



Slika 3. Blok dijagram centralne upravljačke jedinice

3.1. Regulator napona

Regulatori napona u automobilskim sustavima moraju zadovoljiti niz zahtijeva, što uključuje zaštitu od obrnutog polariteta, mogućnost regulacije kod velikih padova napona pri hladnom startu, otpornost na visoke temperature i otpornost na kratkotrajne prenapone u slučaju otpajanja akumulatora. Dvije različite razine napona potrebne su za rad ECU-a. Mikrokontroler, EEPROM i MC33810 napajaju se sa 3.3V, a ostatak komponenata i svi vanjski senzori sa 5V. Jedan regulator napona smanjuje napon akumulatora sa 12V na 5V, a drugi se povezuje na reguliranih 5V i dalje spušta na 3.3V. Dvije su osnovne vrste regulatora, linearni i prekidački. Linearni regulatori čitavu razliku napona pretvaraju u toplinu zbog čega imaju jako malu iskoristivost. Posljedica toga je i značajno zagrijavanje kod velikih razlika ulaznog i izlaznog napona. Zbog toga se za regulaciju 5V koristi prekidačko napajanje (*switch mode power supply*) koje radi na principu prekidanja ulaznog napona velikom frekvencijom čime se postiže znatno veća iskoristivost. Za regulaciju na 3.3V odabran je linearni regulator jer zbog male razlike ulaznog i izlaznog napona zagrijavanja nije problem, a broj pratećih komponenata u odnosu na prekidačko napajanje je znatno manji.

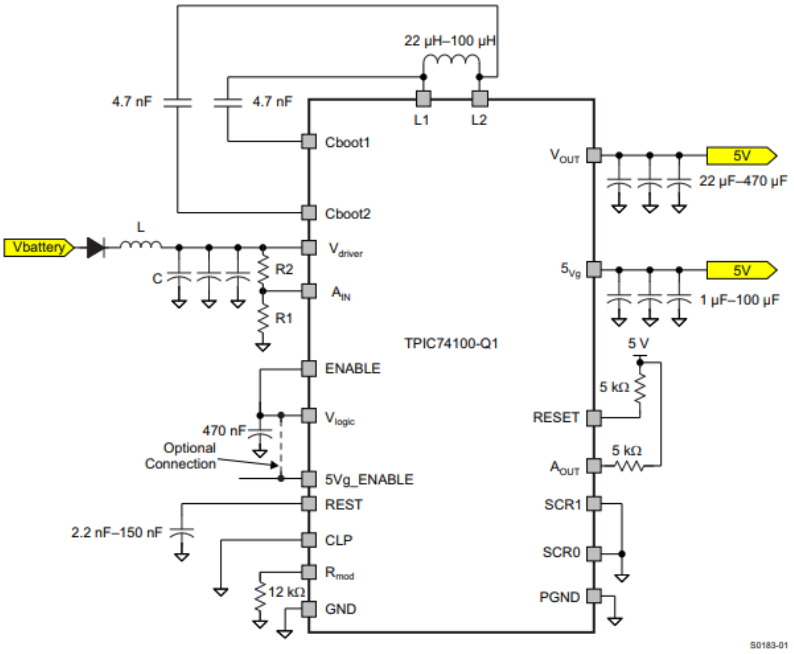
Potrebna struja napajanja određena je na temelju najveće struje svake komponente očitane iz *datasheeta*. Vrijednosti su prikazane u tablici:

Tablica 2. Potrošnja struje pojedinih komponenti

Komponenta	**I_{max}[mA]**
STM32F407	240
MC33810	1
NCV1124	5
EEPROM 25LC640	5
ULN2804	1.4
TPS	2.5
MAP	5
Analog in	20
Digital in	20
Ukupno	299.9

Odabran je regulator TPIC74100-Q1 proizvođača Texas Instruments najveće izlazne struje 1A i dozvoljenog ulaznog napona 1.5V-40V. Za ulazni napon 1.5V regulator može dati 5V na izlazu, ali uz ograničenu struju od 120mA. Veća izlazna struja od potrebne odabrana je

zbog gotovo jednakog broja komponenata i veličine kao i kod slabijih regulatora s time da se ovako dobiva rezerva u slučaju potrebe napajanja dodatnih vanjskih uređaja. Na slici 4. prikazana je preporučena shema spajanja i vrijednosti komponenata.



Slika 4. Shema spajanja prekidačkog regulatora napona TPIC74100

Glavni parametar koji je potrebno odrediti je induktivitet zavojnice regulatora. Zavojnica služi za pohranjivanje energije i izglađivanja valovitosti struje zbog prekidanja napona. Induktivitet se određuje prema sljedećem izrazu danom u uputama proizvođača:

$$L = \frac{(V_{in} - V_{out}) \cdot V_{out}}{f_{sw} \cdot \Delta I_L \cdot V_{IN}} \quad \textbf{(1)}$$

- V_{in} – ulazni napon 13V
- V_{out} – izlazni napon 5V
- f_{sw} – frekvencija prekidanja 400000Hz
- ΔI_L – valovitost izlazne struje, preporuka je $\Delta I_L = 0.2 I_{out} = 0.2 \cdot 0.5 = 0.1A$

Slijedi induktivitet zavojnice:

$$L = \frac{(13-5) \cdot 5}{400000 \cdot 0.1 \cdot 13} = 77\,\mu H \quad \textbf{(2)}$$

Najveća struja prema kojoj se odabire dopuštena struja zavojnice iznosi:

$$I_{L,\max} = I_{out} + \frac{\Delta I_L}{2} = 0.5 + 0.05 = 0.55A \quad \textbf{(3)}$$

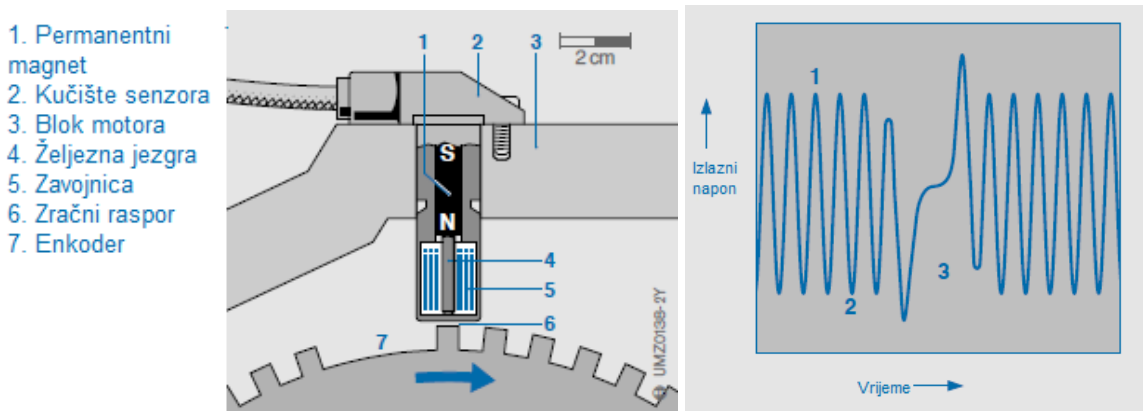
Uz zavojnicu, bitan parametar je i ekvivalentan serijski otpor (ESR) kondenzatora koji zbog visoke frekvencije prekidanja regulatora ima veliki utjecaj na izlaznu valovitost napona. Zato se koriste kondenzatori s niskim ESR-om, a povezivanjem više njih u paralelu, kao što je prikazano na slici , dodatno se snižava otpor i smanjuje valovitost.

Za regulaciju na 3.3V koristi se linearni regulator TLV1117 najveće izlazne struje 800mA. Regulator zahtjeva najmanje 4.3V ulazni napon za regulaciju na 3.3V. U slučaju preopterećenja ili pregrijavanja regulator se automatski gasi da bi spriječio moguće oštećenje.

3.2. Senzori

3.2.1. Senzor kuta zakreta koljenastog vratila

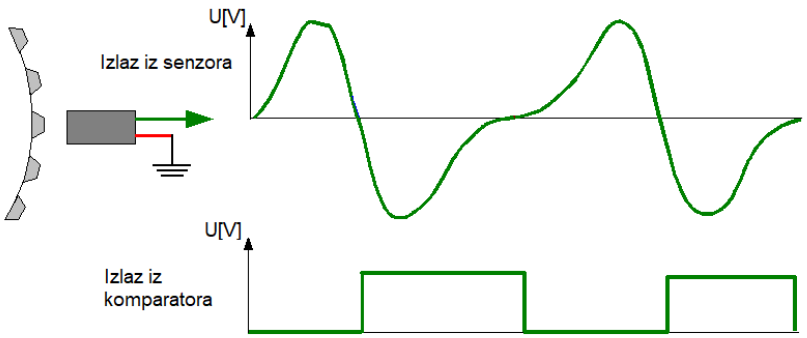
Senzor kuta zakreta koljenastog vratila služi za mjerenje brzine vrtnje i određivanje položaja koljenastog vratila. Najčešće se koriste induktivni i Hallovi, a rjeđe optički senzori. Na bolidu Strix se, kao i na većini motora, na koljenastom vratilu koristi induktivni senzor. Prednosti induktivnih senzora u odnosu na Hallove su manja cijena i otpornost na znatno više temperature zbog toga što nemaju elektroničkih komponenti već samo zavojnicu i magnet. Nedostatak im je što zahtijevaju kompliciranije strujne krugove za kondicioniranje signala i ne mogu raditi pri jako malim brzinama vrtnje.



Slika 5. Senzor zakreta koljenastog vratila[4]

Senzor se postavlja radijalno u odnosu na enkoder napravljen od feromagnetnog materijala uz mali zračni raspor. U senzoru se nalazi jezgra od mekog željeza (Slika 5., 4) omotana zavojnicom (Slika 5., 5) i spojena s permanentnim magnetom (Slika 5., 1). Magnetno polje širi se preko jezgre u enkoder. Iznos magnetnog toka kroz zavojnicu ovisi o tome što se nalazi ispred senzora, zub ili praznina. Magnetski tok kroz zavojnicu biti će veći ako se ispred senzora nalazi zub, a manji u slučaju praznine. Rotacija enkodera na taj način uzrokuje promjenu magnetskog toka zbog čega se u zavojnici inducira izmjenični napon čija amplituda i frekvencija ovise o brzini vrtnje. Napon varira od mV do nekoliko desetaka volti pri višim brzinama vrtnje. Broj zubi na enkoderu ovisi o primjeni, a najčešće upotrebljavani su 24-2, 36-2 i 60-2. U navedenim oznakama -2 znači da 2 zuba na enkoderu nedostaju, a praznina se koristi za sinkronizaciju položaja radilice.

Prije ulaza u mikrokontroler izmjenični napon potrebno je pretvoriti u pravokutni iznosa 0-5V. Strujni krug koji to obavlja u osnovi je komparator koji uspoređuje napon sa senzora s nekim referentnim naponom (oko 0V) te ovisno da li je ulazni napon veći ili manji od referentnog na izlazu daje 0 ili 5V. Slika 6. prikazuje izlazni signal iz senzora i pravokutni signal iz komparatora.

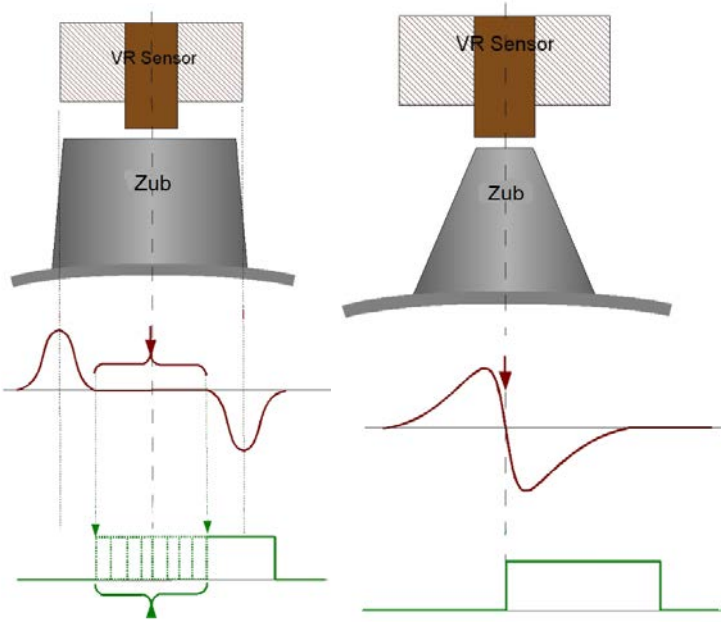


Slika 6. Izlazni signal iz senzora i komparatora[5]

Za očitavanje je teoretski moguće uzeti signal na padajućem ili rastućem rubu pravokutnog signala, ali u praksi se koristi isključivo očitavanje na rastućem rubu tj. kada napon na izlazu iz senzora prelazi sa pozitivnog na negativni napon. Razlog je mnogo oštrija promjena napona (brži prolazak preko 0V) i manje moguće rasipanje prijelaza kod pravokutnog signala.

Veličina željezne jezgre u senzoru mora odgovarati širini zuba na enkoderu da bi se osigurao ispravan oblik signala. Slika 7. (desno) prikazuje ispravan oblik signala za dobro upareni

senzor i širinu zuba, a slika 7. (lijevo) oblik signala za slučaj kada je zub širi od željezne jezgre.



Slika 7. Izlazni signal za različite širine zuba enkodera[5]

Kod prevelike širine zuba prijelaz sa pozitivnog na negativni napon može biti detektiran na bilo kojem dijelu označenom vitičastom zagradom čime se gubi preciznost očitavanja položaja. Na tržištu postoje gotovi integrirani krugovi koji rješavaju kondicioniranje signala iz induktivnih senzora. Zbog njihovih malih dimenzija, niske cijene i relativno malo popratnih komponenata nije isplativo raditi vlastiti strujni krug nego će se iskoristiti neki od gotovih. U sljedećoj tablici navedeni su neki od dobavljivih IC-ova različitih proizvođača:

Tablica 3. Integrirani krugovi za kondicioniranje signala induktivnih senzora

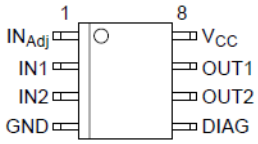
Proizvođač	Model	Broj ulaza	Cijena
Maxim Integrated	MAX9927	2	4,81
ON Semiconductors	NCV1124	2	1,93
Texas Instruments	LM1815	1	2,83

Glavna razlika MAX9927 i LM1815 u odnosu na NCV1124 je što imaju adaptivnu referentnu vrijednost komparatora koja ovisi o naponu na senzoru. Pri višim brzinama vrtnje, kada je napon na senzoru veći, referentna vrijednost na komparatoru također je veća zbog čega je

potreban viši napon da bi komparator promijenio vrijednost na izlazu. Time se dobije veća imunost na smetnje u signalu.

LM1815 je odbačen zbog relativno velikih dimenzija i cijene, kada se uzme u obzir da ima samo jedan ulazni kanal. Od preostala dva odabran je NCV1124 zbog znatno manjeg broja potrebnih pratećih komponenti i manje cijene, a kako je kabel od senzora oklopljen cijelom dužinom, smetnje neće predstavljati problem.

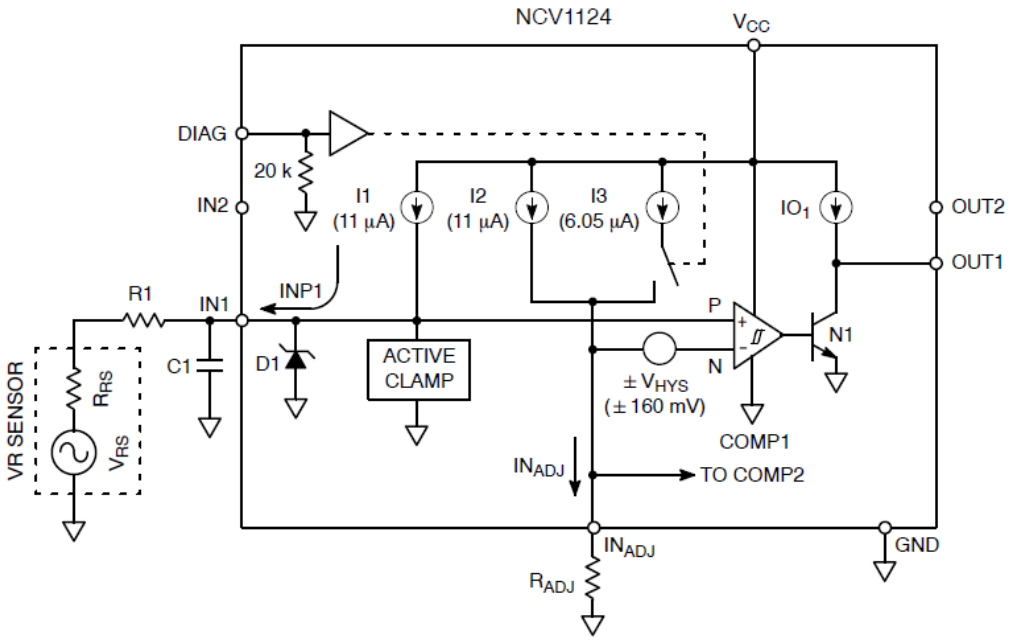
NCV1124 je integrirani krug s dva kanala koji signal dobiven iz induktivnog senzora pretvara u pravokutni signal. Dostupan je u pakiranju SO-8 prikazanom na slici 8., okvirnih dimenzija 5mm x 6,2mm.



Slika 8. Integrirani krug za kondicioniranje signala induktivnih senzora-NCV1124

Da bi IC radio kako je predviđeno, potrebno je proučiti princip rada i odabrati ispravne vrijednosti pratećih komponenata. Induktivni (*VR-variable reluctance*) senzor ima dva kontakta, jedan se spaja na masu, a drugi preko otpornika R1 na ulaz IN1. Otpornik R1 i kondenzator C1 tvore *low-pass* filter za uklanjanje visokofrekventnih smetnji. Otpornik R1 također ograničava najveću struju koja može poteći i ne smije prekoračiti 12mA zbog aktivne hvataljke koja ograničava napon na 5V. Iz najvećeg napona senzora i dozvoljene struje može se izračunati vrijednost otpornika R1. Za najveći napon senzora uzeta je vrijednost V_{RSmax}=140V :

$$R_1 = \frac{V_{RSmax}}{I_{CLMP}} = \frac{140}{0.012} = 11666\Omega = 12k\Omega \qquad \textbf{(4)}$$



Slika 9. Blok dijagram integriranog kruga za kondicioniranje signala induktivnih senzora-NCV1124

Kada na senzoru nema napona (V_{RS}=0), IN1 se nalazi na naponu koji je određen izvorom struje I_1 i otporima R_1 i R_{RS} (otpor senzora):

$$U_{IN1} = I_1(R_1 + R_{RS}) \quad \textbf{(5)}$$

Napon IN1 jednak je naponu na komparatoru V_P. Napon V_N na komparatoru određen je strujom I_2 (ako je dijagnostički mod isključen) i otpornikom $R_{ADJ,}$ plus ili minus napon histereze V_{HYS}:

$$V_N = I_2 \cdot R_{ADJ} \pm V_{HYS} \quad \textbf{(6)}$$

Ako je $V_N > V_P$ izlaz komparatora je logički 0, tranzistor N1 je ugašen, a OUT1 jednak Vcc. U suprotnom izlaz komparatora je logički 1, tranzistor N1 provodi, a OUT1 je spojen na masu. Otpornikom R_{ADJ} mijenja se napon V_N čime se utječe na potreban napon V_P da bi komparator promijenio izlaznu vrijednost. Veća vrijednost otpora povećava napon senzora potreban za promjenu izlaza komparatora i tako povećava otpornost na smetnje. S druge strane otpor mora biti dovoljno malen da se mogu detektirati male promjene napona senzora kod pokretanja

motora. Ako se prag komparatora želi postaviti na 0V ± napon histereze vrijednost otpora R_{ADJ} mora biti jednaka:

$$R_{ADJ} = R_1 + R_{RS} \quad \textbf{(7)}$$

Prag komparatora može se izračunati prema sljedećem izrazu:

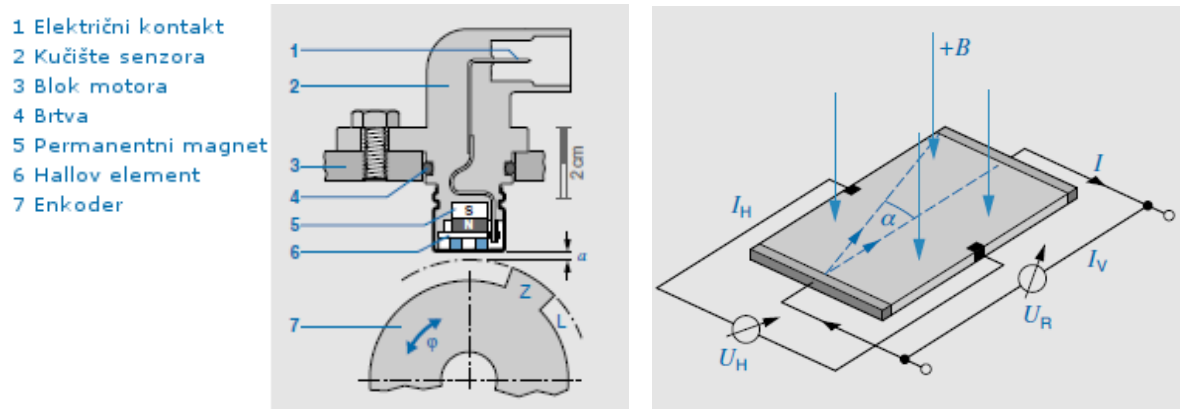
$$V_{RS(+TRP)} = I_2 \cdot R_{ADJ} - I_1(R_1 + R_{RS}) + V_{HYS} \quad \textbf{(8)}$$

Za otpornik R_{ADJ} postaviti će se promjenjivi otpornik kako bi se naknadno mogao mijenjati prag komparatora prema korištenom senzoru.

3.2.2. Senzor kuta zakreta bregastog vratila

Senzor kuta zakreta bregastog vratila koristi se zajedno sa senzorom na koljenastom vratilu za potpuno definiranje faze pojedinog cilindra. Jednocilindrični motori i motori sa parnim brojem cilindara mogu raditi bez senzora na bregastom vratilu tako da u svakom cilindru iskra preskače dva puta u ciklusu. Razlog je što nije moguće odrediti je li pojedini cilindar na paljenju ili izmjeni radnog medija.

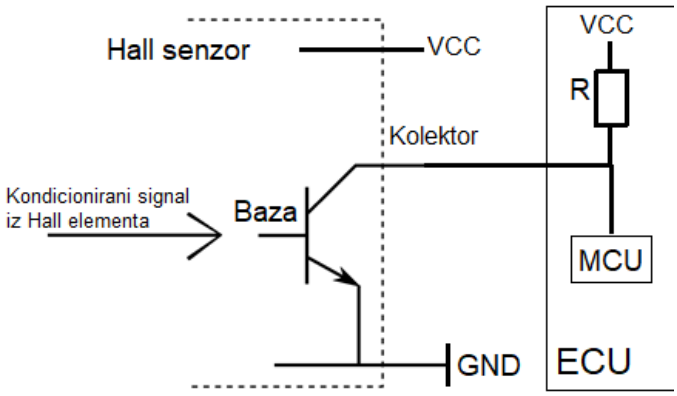
Najčešće korišteni senzori na bregastim vratilima su Hallovi senzori. Za razliku od induktivnih senzora oni zahtijevaju napajanje (5 ili 12V), izlazni napon im ne ovisi o brzini vrtnje, a poziciju mogu detektirati i u mirovanju.



Slika 10. Hallov davač [4]

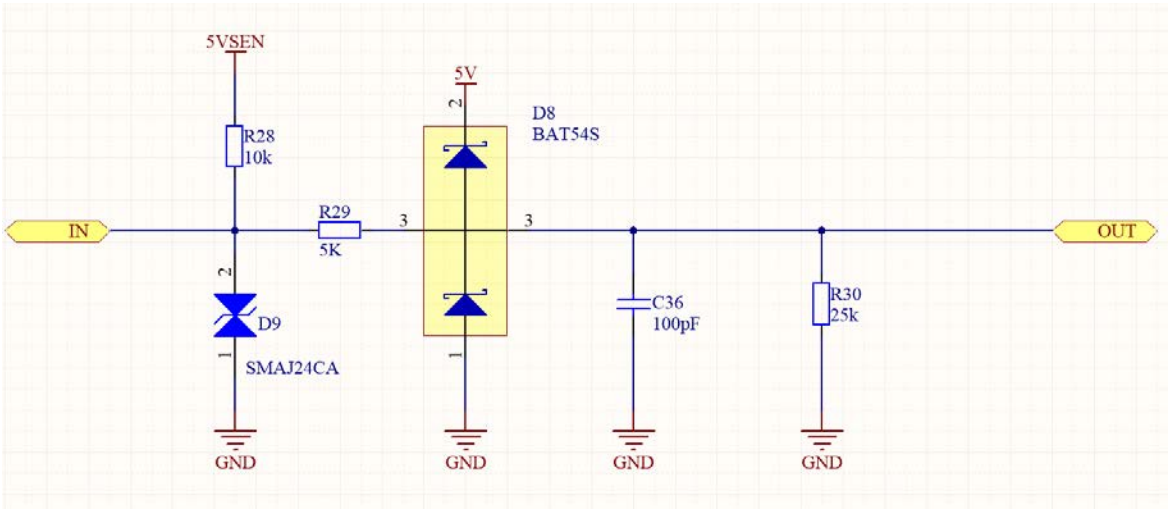
Kao što ime kaže, ovi senzori koriste Hallov efekt. Enkoder od magnetičnog materijala rotira sa bregastim vratilom. Hallov element nalazi se u kućištu senzora između enkodera i permanentnog magneta koji stvara magnetno polje okomito na Hallov element. Kada se zub

na enkoderu nađe ispred senzora promjeni se jakost magnetnog polja kroz element što uzrokuje skretanje elektrona (kut α na slici 10.) koji se gibaju uzdužno u elementu, a time i malu promjenu napona U_H. Integrirani krug u senzoru pomoću te promjene napona pali i gasi tranzistor (Slika 11.) na čiji je kolektor spojen *pull-up* otpornik R. Kada tranzistor provodi, na kolektoru je napon jednak 0V, a u suprotnom 5V ili 12V, ovisno o naponu na koji je spojen otpornik. Na taj način mikrokontroler dobiva jasan digitalni signal.



Slika 11. Osnovna shema spajanja Hallovog davača

Izlaz iz senzora spaja se na mikrokontroler preko zaštitnog kruga na slici 12. koji se ujedno koristi i na svim ostalim digitalnim ulazima. Svrha zaštite je sprječavanje prolaza visokog napona zbog nepravilnog spajanja ili tranzijentnih pojava koje mogu uzrokovati prenapone ili negativne napone. Dozvoljeni napon na digitalnim ulazima je 5V pa je potrebno osigurati da njegov iznos nikada ne prijeđe tu vrijednost.



Slika 12. Zaštitni krug digitalnih ulaznih kanala

Signal sa senzora, preko jednog od pinova na konektoru, dolazi u ECU te se spaja na ulaz u zaštitni krug (*IN*). Zaštitni krug je osmišljen tako da je moguće trajno spojiti do 24V na ulaz bez da dođe do oštećenja mikrokontrolera. Dvosmjerna TVS dioda (*transient voltage suppresion* dioda) D9 štiti od tranzijentnih skokova napona. Ako je napon na ulazu ispod probojnog napona diode ona ne propušta struju i ponaša se kao otvoreni krug. Ako napon skoči iznad probojnog napona, dioda počne voditi kako bi održala napon na toj razini. TVS diode ne mogu zaštiti ulaz od dugotrajnih prenapona te se za tu svrhu koristi par Schottky dioda D8 sa otpornikom R29 u seriji. Otpornik ograničava struju koja teče kroz diodu u slučaju da napon naraste iznad 5V ili padne ispod 0V. Za slučaj da se na ulazu pojavi 24V struja koja teče kroz diodu jednaka je:

$$I_{\mathrm{D8}} = \frac{U_{\mathrm{in}} - (5 + V_{\mathrm{F}})}{R_{29}} - \frac{5 + V_{\mathrm{F}}}{R_{30}} = \tag{9}$$

$$= \frac{24 - (5 + 0.4)}{5000} - \frac{5 + 0.4}{25000} = 3.50\mathrm{mA} < I_{\max} = 200\mathrm{mA}$$

gdje je V_{F} pad napona na diodi, a $I_{\max}$ najveća dozvoljena kontinuirana struja diode.

Dok je napon u granicama 0-5V gornja dioda sprječava tok struje sa 5V, a donja tok struje na masu. Ako napon naraste iznad 5V gornja dioda slobodno provodi u tom smjeru te održava ulazni napon na 5V uvećan za pad napona na diodi. Kondenzator C36 s otpornikom R28 i R29 tvori *low pass* filter koji služi za filtriranje visokih frekvencija. Vremenska konstanta kod punjenja kondenzatora tj. vrijeme potrebno da napon naraste od 0 do 63.2% konačne vrijednosti jednaka je:

$$\tau = RC = 14700 \cdot 100 \cdot 10^{-12} = 1.47 \cdot 10^{-6}\mathrm{s} \tag{10}$$

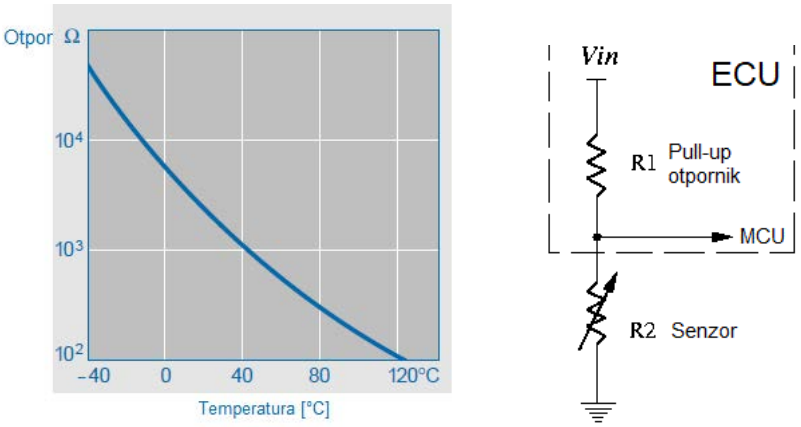
Ova vremenska konstanta odbrana je zato što mikrokontroler logički 1 na ulazu detektira na naponu od oko $0.6 \cdot V\mathrm{cc}$ što znači da bi kašnjenje na odskočnu funkciju iznosilo oko $1.5\mu\mathrm{s}$. Ako se uzme u obzir da 1° zakreta KV pri $8000\mathrm{min}^{-1}$ traje 21µs, kašnjenje od 1.5µs je prihvatljivo. OUT se spaja na jedan od digitalnih ulaza na mikrokontroleru.

3.2.3. Senzori temperature

Postoji čitav niz različitih vrsta temperaturnih senzora ovisno temperaturnom rasponu i primjeni. Na motorima se za mjerenje nižih temperatura(do 150°C) najčešće koriste termistori tj. otpornici koji mijenjaju otpor u ovisnosti o temperaturi. Dvije su vrste termistora, NTC (*negative temperature coefficient*) kojima s porastom temperature pada otpor i PTC (*positive temperature coefficient*) kojima otpor raste porastom temperature. Gotovo uvijek se za mjerenje temperature koriste NTC otpornici. Sam termistor prikazan je na slici 13.(lijevo) i nalazi se u kućištu senzora koje može biti otvoreno ili zatvoreno. Svaka nožica termistora spojena je direktno na svoj kontakt na konektoru senzora.

Slika 13. NTC otpornik(lijevo), senzor temperature vode(sredina), senzor temperature zraka(desno)

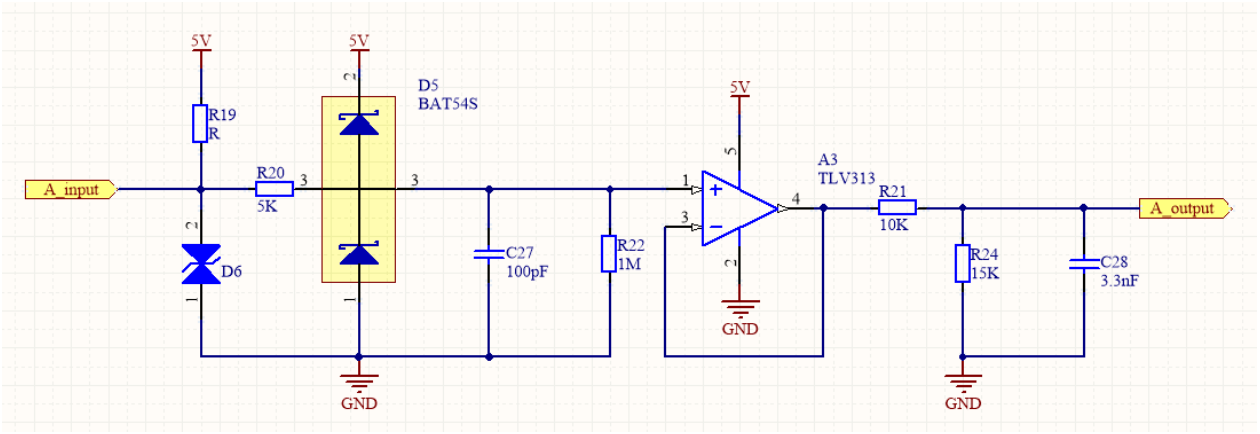
Ovi senzori koriste se za mjerenje temperature zraka, rashladne tekućine, temperature ulja i temperature goriva. Tipična karakteristika NTC otpornika vidi se na slici 14. (lijevo). NTC otpornici deklariraju se prema iznosu otpora na 25°C („NTC 2.2k“ znači da senzor ima otpor 2200Ω na 25°C).



Slika 14. Tipična karakteristika NTC otpornika(lijevo)[4], osnovna shema spajanja NTC otpornika (desno)

Otpor nije moguće izravno mjeriti mikrokontrolerom već je potrebno promjenu otpora na neki način pretvoriti u promjenu napona. To se izvodi dodavanjem otpornika R1 (Slika 14. desno) prema 5V koji sa senzorom tvori naponsko dijelilo. Promjenom otpora na senzoru mijenja se izlazni napon prema mikrokontroleru prema izrazu (11). U programu je pohranjena karakteristična krivulja pomoću koje se iz napona izračunava temperatura.

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2} \tag{11}$$

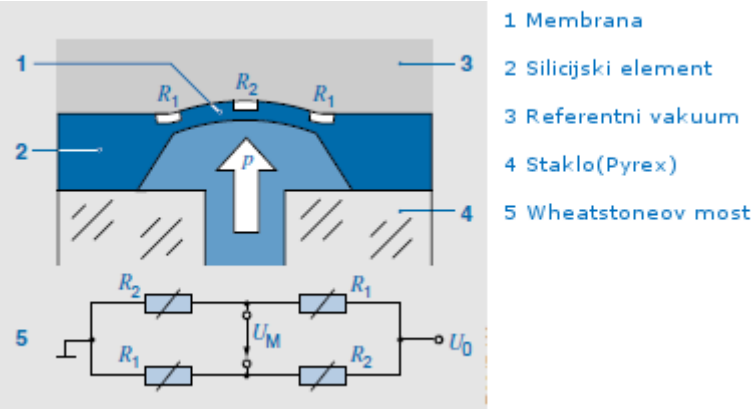


Slika 15. Zaštitni krug analognih ulaznih kanala

Princip rada TVS i Schottky dioda identičan je kao i kod digitalnih ulaza, a *pull-up* otpornik R19 koristi se samo kod senzora koji mijenjaju otpor tj. ne daju napon na izlazu. Kod analognih ulaza koristi se i operacijsko pojačalo A3 koje dodatno štiti ulazne kanale s analogno-digitalnim pretvaračem zbog njihove veće osjetljivosti na skokove napona. Operacijsko pojačalo u ovom slučaju radi kao naponsko slijedilo tj. ulazni napon jednak je izlaznom naponu. Najveći dozvoljeni napon na ulazima s ADC-om iznosi 3.3V, a sa 5V se snižava pomoću naponskog dijelila kojeg tvore otpornici R21 i R24. Otpornik R21 i kondenzator C28 čine niskopropusni filtar koji se podešava ovisno o potrebama odziva pojedinog senzora.

3.2.4. Senzor tlaka u usisnom kolektoru

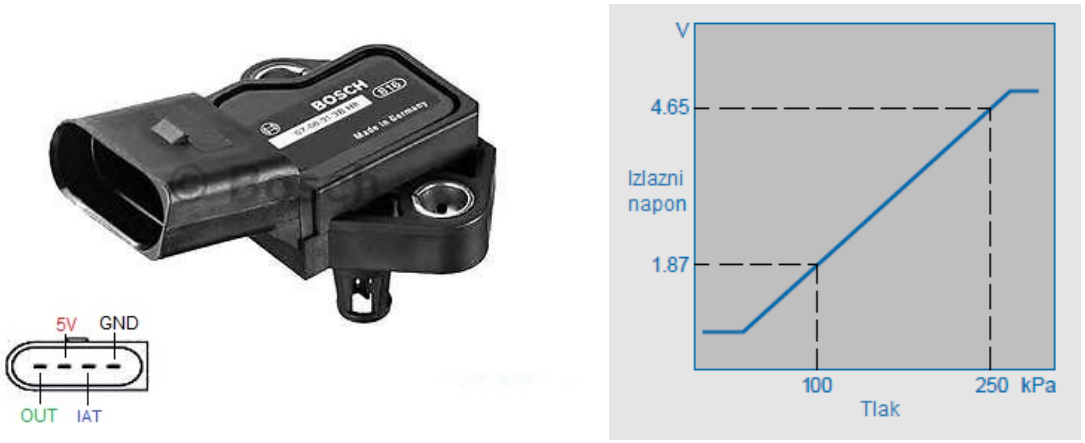
Ovaj senzor mjeri apsolutni tlak u usisnom kolektoru i uspoređuje ga s referentnim vakuumom. Poznavanjem apsolutnog tlaka i temperature zraka može se procijeniti maseni protok zraka u cilindar. Glavni dio senzora je silicijski čip (Slika 16., 2) u kojem je mikro mehanički ugravirana tanka membrana(1) na kojoj se nalaze četiri deformacijska otpornika (R1,R2) čiji se otpor mijenja kao funkcija mehaničkog naprezanja uslijed djelovanja tlaka.



Slika 16. Konstrukcija MAP senzora[4]

Mjerni element smješten je u kućištu u kojem se nalazi referentni vakuum s jedne strane, a mjereni tlak sa druge strane. Membrana senzora deformira se manje ili više (10...1000µm) ovisno o mjerenom tlaku. Mjerni otpornici na silicijskom čipu raspoređeni su tako da se kod deformacije membrane otpor jednog para smanjuje, a drugog povećava. Otpornici tvore Wheatstoneov most (5) kod kojeg promjena otpora dovodi do promjene omjera napona , a time i mjerenog napona U_M. Wheatstoneov most omogućuje veću osjetljivost senzora jer je mjereni napon veći nego što bi bio u slučaju korištenja zasebnih otpornika.

Elektronika za kondicioniranje signala integrirana je unutar kućišta senzora i koristi se za pojačavanje mjerenog napona U_M, kompenzaciju temperaturnih utjecaja i linearizaciju ovisnosti napona i tlaka. Tako se na izlazu iz senzora dobiva napon od 0-5V koji linearno raste porastom tlaka (Slika 17, desno). ECU iz vrijednosti napona izračunava tlak prema karakteristici koju daje proizvođač senzora.



Slika 17. MAP senzor i njegova izlazna karakteristika[4]

Na slici 17. lijevo prikazan je MAP senzor koji će se koristiti na bolidu Strix, a može se pronaći na većini motora iz VAG grupacije. U istom kućištu sa MAP senzorom nalazi se i senzor temperature usisnog zraka zbog čega konektor ima četiri kontakta: 5V (napajanje senzora), GND, signal i izlaz senzora temperature. Postoji nekoliko verzija senzora ovisno jesu li ugrađivani na motore sa prednabijanjem ili bez, a razlikuju se po mjernom području. Senzori za motore bez prednabijanja najčešće imaju mjerno područje u rasponu od 15kPa do 120kPa te za razliku od senzora sa širim mjernim područjem imaju veću rezoluciju (veću promjenu napona za istu promjenu tlaka).

3.2.5. Lambda sonda

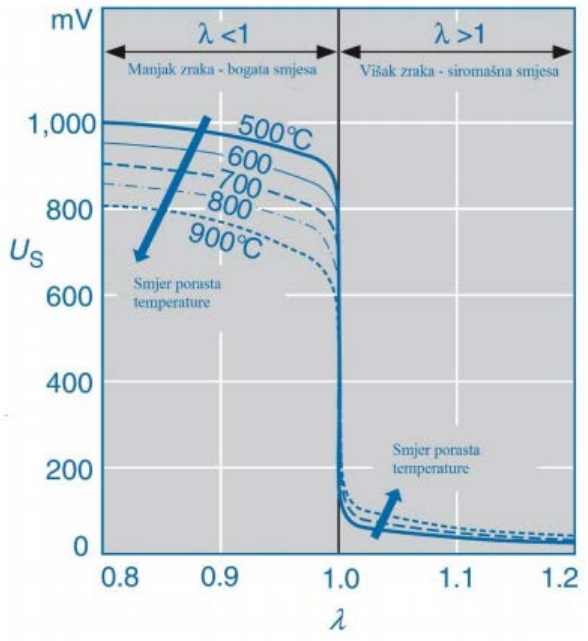
Lambda sonda koristi se za određivanje faktora pretička zraka mjerenjem koncentracije kisika u ispušnim plinovima. Smještena je u ispušnoj cijevi nakon kolektora gdje na nju nastrujavaju ispušni plinova iz motora. Prema mjernom području dijele se na uskopojasne i širokopojasne. Uskopojasne lambda sonde precizne su u vrlo uskom području faktora pretička zraka oko λ=1 i koriste se za regulaciju kod motora sa katalizatorima. Širokopojasne lambda sonde omogućuju mjerenje faktora pretička zraka u znatno širem području $0.7<\lambda<\infty$ zbog čega se mogu koristiti kod motora koji rade sa siromašnim ili bogatim smjesama. Na motorima namijenjenim za utrke koriste se isključivo širokopojasne lambda sonde jer većinu vremena rade sa bogatom smjesom (λ=0.85) kod koje se najčešće razvija najveća snaga.

Princip rada

Širokopojasna lambda sonda u osnovi je uskopojasna lambda sonda kojoj je dodan transportni članak, zato će se najprije opisati princip rada uskopojasne lambda sonde. Njen rad temelji se na principu elektrokemijskog članka (Nernstov članak) koji se sastoji od keramičkog elementa (ZrO_2)(Slika 18., 1) obloženim tankim slojem platine (Slika 18., 2). Jedna strana keramičkog elementa izložena je ispušnim plinovima, a druga okolnom zraku.

Slika 18. Shematski prikaz uskopojasne lambda sonde[4]

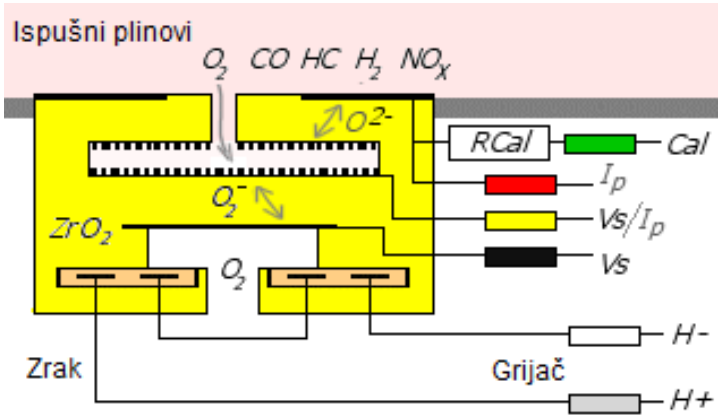
Važno svojstvo cirkonijevog oksida je da pri temperaturama višim od 350°C postaje vodljiv za ione kisika te ako se u ispušnim plinovima pojavi manjak kisika u odnosu na referentnu vrijednost dolazi do prolaska iona s jedne na drugu stranu i pojavljuje se razlika potencijala.
Sloj platine služi kao elektroda koja prenosi naboj do vodiča koji vode iz senzora. U slučaju bogate smjese, napon na elektrodama iznosi oko 800-1000mV, a u slučaju siromašne smjese manje od 100mV. Kod stehiometrijske smjese napon iznosi oko 450mV, ali zbog jako strme karakteristike varira između 100-900mV. Promjenom temperature keramičkog elementa mijenja se njegova vodljivost iona kisika, a time i sama izlazna karakteristika senzora (Slika 19.). Kako je optimalna radna temperatura senzora oko 600°C, u sondama se obično nalazi grijač koji ubrzava postizanje radne temperature ili održava radnu temperaturu u slučaju kada temperatura ispušnih plinova nije dovoljno visoka.



Slika 19. Izlazna karakteristika uskopojasne lambda sonde[1]

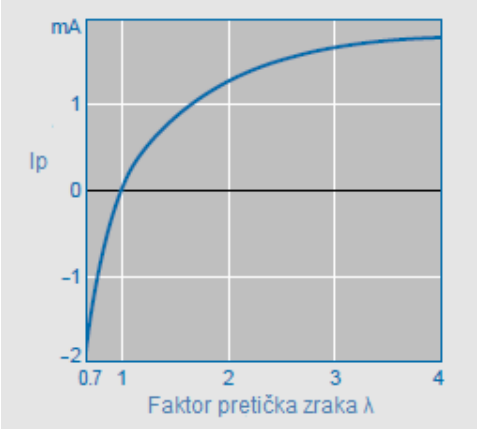
Upravljanje širokopojasnom lambda sondom

Širokopojasna lambda sonda sastoji se od Nernstovog članka, identičan onome kod uskopojasne lambda sondi, i transportnog članka koji služi za prijenos kisika prema mjernom članku. Nernstov članak s jedne je strane izložen referentnoj komori, a s druge difuzijskoj komori. Difuzijska komora izložena je ispušnim plinovima preko uskog prolaza koji dozvoljava ograničen prolazak molekula ispušnih plinova. Propuštanjem struje kroz difuzijski članak moguće je transportirati ione kisika iz ili u difuzijsku komoru.



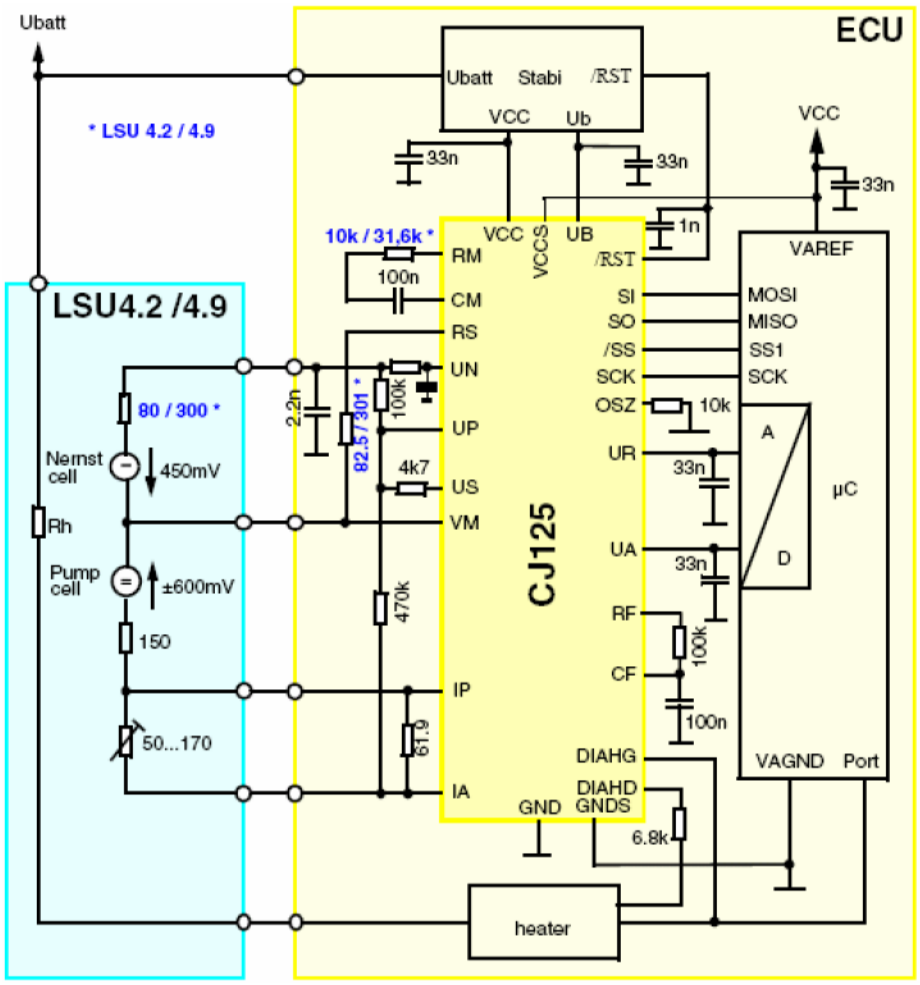
Slika 20. Shematski prikaz širokopojasne lambda sonde[6]

Pomoću Nernstovog članka mjeri se koncentracija kisika u difuzijskoj komori koja upravljačkoj elektronici koristi za regulaciju struje kroz transportni članak. Ako je smjesa siromašna(λ>1) napon na članku jednak je približno 100mV pa upravljačka elektronika propušta struju (*I*p) kroz transportni članak kako bi se izbacio dio iona kisika i postigla stehiometrijska smjesa. Kod bogate smjese mijenja se smjer struje kroz difuzijski članak pa se ioni kisika prebacuju iz ispušnih plinova u difuzijsku komoru. Faktor pretička zraka određuje se iz struje koja je potrebna da bi se u difuzijskoj komori održala stehiometrijska smjesa (Slika 21).



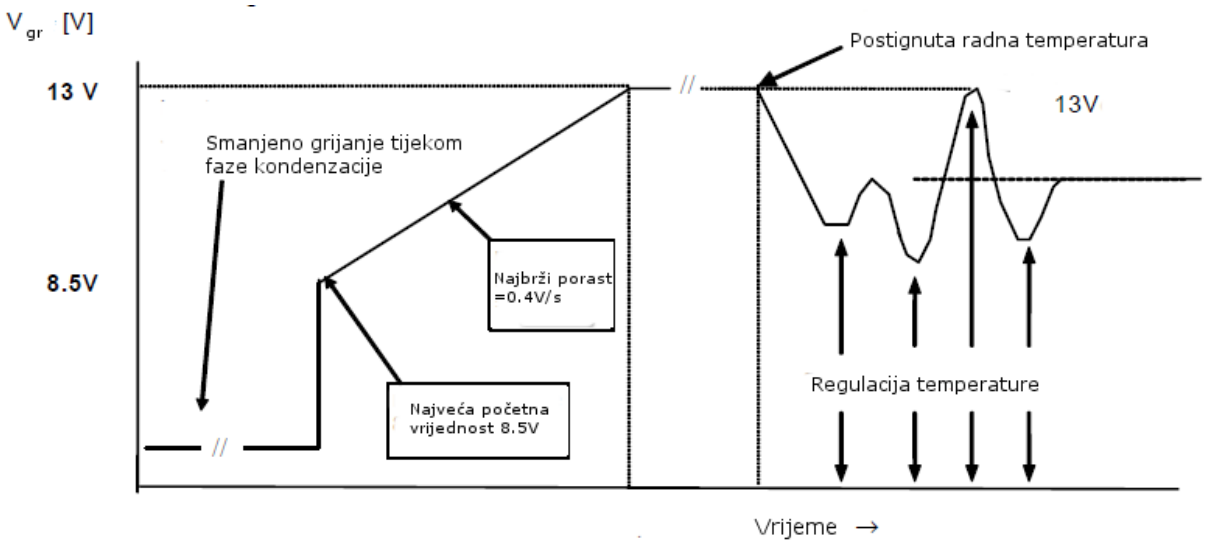
Slika 21. Ovisnost struje transportnog članka o faktoru pretička zraka[1]

Širokopojasna lambda sonda za razliku od uskopojasne zahtjeva relativno složen kontroler kojemu je zadatak održavati optimalnu temperaturu senzora i stehiometrijsku smjesu u difuzijskoj komori. Na tržišu postoje gotovi kontroleri koji se brinu o regulaciji svih parametara potrebnih za ispravan rad lambda sonde, a na izlazu daju napon 0-5V linearno ovisan o faktoru pretička zraka. Nedostatci takvih kontrolera su relativno velike dimenzije jer većinom nisu namijenjeni za ugradnju na tiskanu pločicu već se nalaze u vlastitom kućištu, visoka cijena i upitna kvaliteta onih povoljnijih. U prvoj verziji ECU-a koristit će se gotovi kontroler SLC OEM koji najbolje zadovoljava tražene zahtjeve i moguće ga je implementirati na tiskanu pločicu. Drugo rješenje je integrirani krug od Boscha namijenjen za upravljanje njihovim LSU4.2 i 4.9 lambda sondama. Njime se rješava regulacija struje kroz transportni članak i mjerenje otpora Nernstovog članka dok je regulaciju struje grijača potrebno izvesti sa zasebnim mikrokontrolerom.



Slika 22. Shema spajanja upravljačke jedinice širokopojasne lambda sonde

Shema spajanja upravljačke jedinice CJ125 zajedno sa shemom lambda sonde prikazana je na slici 22. Za ispravan rad sonde najprije je potrebno postići radnu temperaturu. Struja kroz grijač regulira se PWM-om frekvencije veće od 20Hz. U početnoj fazi zagrijavanja kod hladnog motora kada dolazi do pojave kondenziranja vode u ispušnoj cijevi napon na grijaču održava se na 2V (oko 15% DC za napon 13V). Nakon toga se trenutno diže do najviše 8.5V i povećava za 0.4V/s do postizanja napona akumulatora.



Slika 23. Upravljanje zagrijavanjem lambda sonde

Kada se postigne radna temperatura prelazi se na regulaciju napona s ciljem održavanja što manjeg odstupanja od optimalne temperature. Temperatura se mjeri indirektno, mjerenjem otpora Nernstovog članka na način da se kroz njega propušta izmjenična struja jakosti do 250μA i frekvencije 1-4kHz (Slika 22., U_N). Poznavanjem struje i pada napona moguće je izračunati otpor Nernstovog članka. Za LSU4.2 sondu otpor na radnoj temperaturi (780°C) iznosi 80Ω, a za LSU4.9 300Ω. Iznos otpora dobiva se na izlazu U_R kao napon, a otpor se izračunava prema sljedećem izrazu:

$$R_i = \frac{U_R - 0.294}{0.00245} \qquad \textbf{(12)}$$

Mikrokontroler pomoću PID regulatora ima zadatak držati otpor u što užim granicama jer odstupanje temperature Nernstovog članka od optimalne uzrokuje grešku mjerenja(4% za odstupanje 100°C).

Samo mjerenje faktora pretička zraka svodi se na mjerenje struje kroz transportni članak koja je potrebna za održavanje 450mV na Nernstovom članku tj. stehiometrijske smjese u difuzijskoj komori. Ako je napon na Nernstovom članku različit od 450mV, propušta se struja iz I_A preko otpornika R=61.9Ω kako bi se u difuzijskoj komori održala stehimoetrijska smjesa. Pad napona na otporniku koristi se za izračunavanje jakosti struje. Svaka lambda sonda dodatno ima tvornički kalibriran otpornik s kojim se osigurava da je njezina karakteristika

(ovisnost struje i faktora pretička zraka) uvijek ista. Na izlazu U_A dobiva se napon iz kojeg se izračunava struja prema izrazu:

$$I_p = \frac{U_A - 1,5}{(61.9/1000)\cdot v} \quad \textbf{(13)}$$

gdje je *v* osjetljivost koju je moguće podesiti u CJ125. Za *v*=8 mjerno područje je $0.8<\lambda<\infty$, a za *v*=17 mjerno područje je $0.65<\lambda<\infty$.

3.2.6. Senzor stupnja prijenosa

Senzor stupnja prijenosa mjeri kut zakreta bubnja pomoću kojeg se iz *lookup* tablice određuje trenutni stupanj prijenosa. Za tu svrhu koristi se beskontaktni rotacijski senzor pozicije koji radi na principu Hallovog efekta. Za razliku od klasičnog potenciometra, ovaj ima znatno dulji vijek trajanja i manje izražene šumove u signalu. Izlazni signal nalazi se u rasponu 0.5-4.5V i linearno je ovisan o kutu zakreta.

Podatak o stupnju prijenosa, osim za prikaz vozaču, koristi se za izračunavanje brzine vožnje iz trenutne brzine vrtnje i stupnja prijenosa te za onemogućavanje prebacivanja iz prvog u neutralni stupanj prijenosa dok se vozilo kreće.

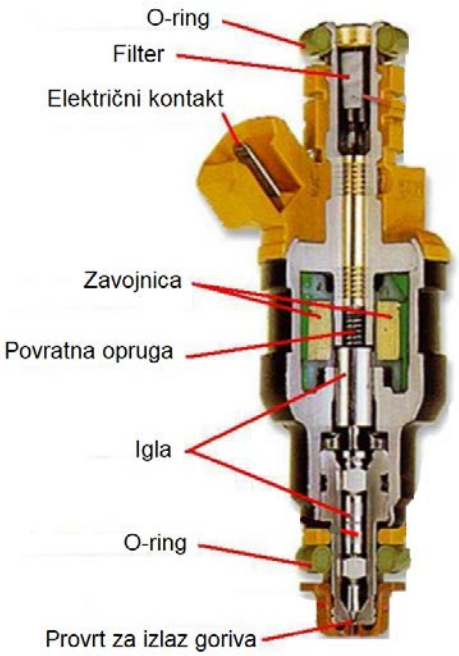
3.2.7. Ulazi opće namjene

U ovu skupinu pripadaju svi ostali ulazi sa raznih prekidača i senzora kao što su prekidač pedale kvačila, prekidači za mijenjanje stupnja prijenosa, senzor brzine vrtnje kotača i ostalo. Spajaju se na identične digitalne i analogne ulazne krugove kao i svi prethodno opisani senzori.

3.3. Aktuatori

3.3.1. Brizgaljka goriva

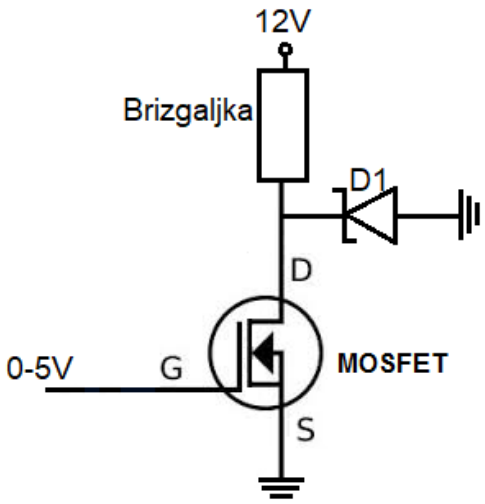
Brizgaljka goriva u osnovi je elektromagnetski ventil kojemu je zadatak precizno doziranje goriva u obliku fino raspršenog mlaza u usisnu granu. U ovome dijelu govorit će se o brizgaljkama korištenim kod niskotlačnih sustava ubrizgavanja goriva u usisnu granu s radnim tlakovima od 3-4bara. Gorivo iz razdjelne cijevi ulazi u brizgaljku i prolazi kroz filter koji sprječava prolazak sitnih čestica koje bi mogle oštetiti ili začepiti uske prolaze unutar brizgaljke. Opruga (Slika 24.) pritišče iglu brizgaljke na sjedište čime se zatvara provrt za izlaz goriva. U sredini brizgaljke nalazi se zavojnica čiji su krajevi spojeni s kontaktima na konektoru. Propuštanjem struje kroz zavojnicu dolazi do stvaranja magnetskog polja koje privlači magnet, podiže iglu sa sjedišta i otvara provrt za izlaz goriva.



Slika 24. Presjek brizgaljke goriva

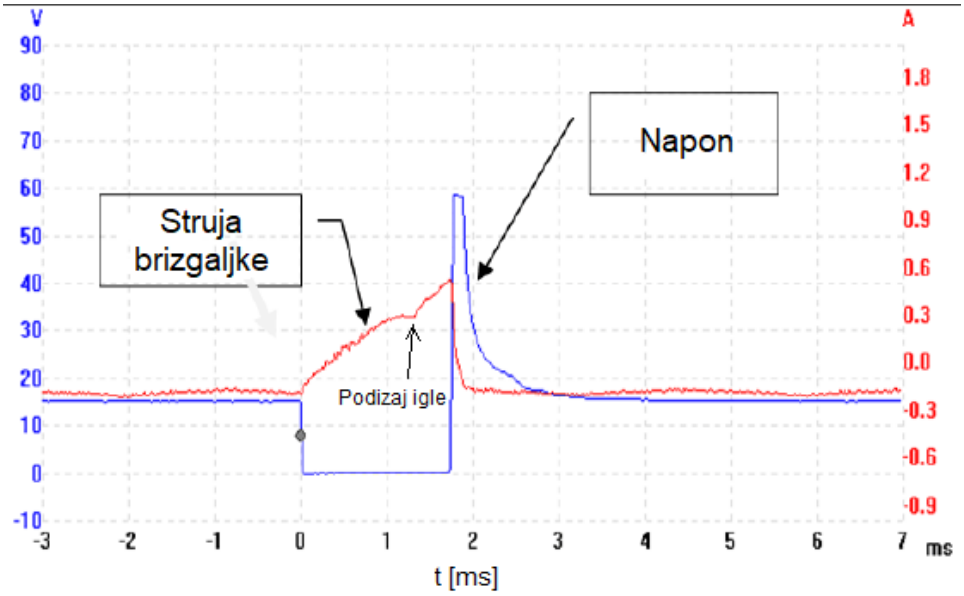
Sve brizgaljke kod sustava ubrizgavanja u usisnu granu upravljaju se izravno sa 12V. Mogu se podijeliti na dvije glavne skupine s obzirom na način upravljanja, a to su brizgaljke s visokom impedancijom (*high-Z*) i brizgaljke s niskom impendancijom (*low-Z* ili *peak & hold* brizgaljke). Brizgaljke s niskom impedancijom(otpor 2-4Ω) upravljaju se na način da se u početnoj fazi spajaju izravno na 12V s ciljem što bržeg porasta struje u zavojnici, a time i

bržeg podizanja igle. Nakon toga napon na brizgaljci smanjuje se PWM-om na oko 30% nazivnog napona kako bi se struja ograničila na razinu koja neće uzrokovati pregrijavanje brizgaljke. Kada bi napon čitavo vrijeme bio 12V, struja kroz zavojnicu iznosila bi oko 4A što bi dovelo do pregrijavanja i oštećenja brizgaljke. Ovaj tip upravljanja koristio se u prošlosti kod brizgaljki s velikim protocima jer je omogućavao brže otvaranje i preciznije doziranje goriva kod kratkih otvorenosti brizgaljki. Suvremene *high-Z* brizgaljke zbog užih tolerancija i znatno lakših pokretnih dijelova omogućuju precizno doziranje i kod najvećih protoka (iznad 2000ml/min) te su znatno jednostavnije za upravljane. Na bolidu Strix koristi se Bosch EV14 brizgaljka, najnoviji model Bosch-ovih *high-Z* brizgaljki. Zbog većeg otpora zavojnice (12-15Ω) najveća struja pri 12V ograničena je na približno 1A pa ih je moguće čitavo vrijeme držati na nazivnom naponu. Osnovni strujni krug potreban za upravljanje ovom vrstom brizgaljke prikazan je na slici 25. Jedan kontakt spojen je izravno na 12V, a drugi na odvod (*drain*) MOSFET-a. Kada je napon na upravljačkoj elektrodi (*gate*) jednak 0V MOSFET ne provodi pa je i struja kroz zavojnicu brizgaljke jednaka nuli.



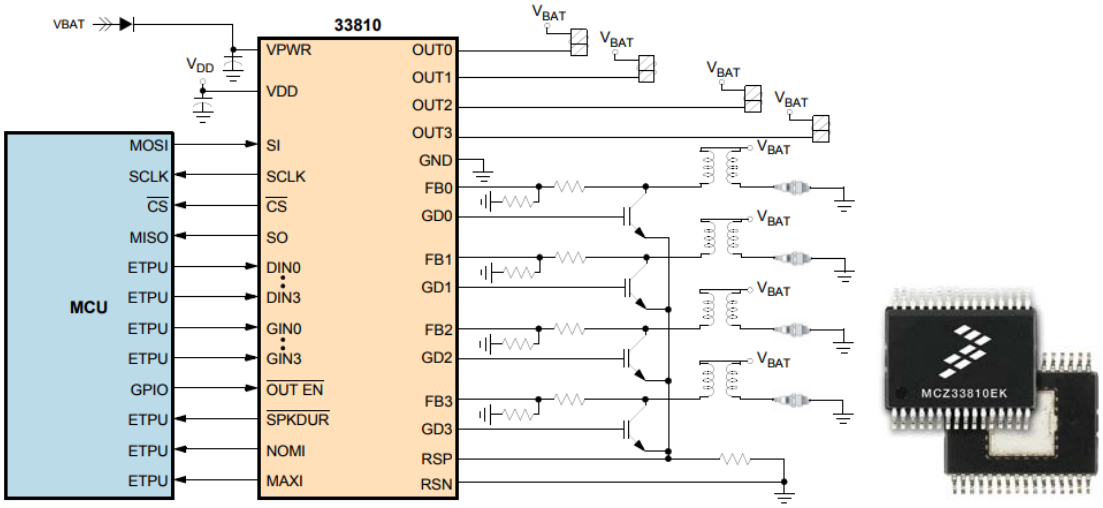
Slika 25. Osnovna shema spajanja brizgaljke

Ako se na upravljačku elektrodu dovede napon (5V za *logic-level* MOSFET-e), MOSFET provodi i kroz zavojnicu počinje teći struja. Napon na uvodu (*drain*) MOSFETA trenutno pada na približno 0V, a struja raste brzinom koja ovisi o induktivitetu i otporu zavojnice.



Slika 26. Napon i struja brizgaljke

Porastom struje u zavojnici proporcionalno raste jakost magnetnog polja koje privlači magnet na vrhu igle brizgaljke. Kada jakost magnetnog polja naraste dovoljno da privlačna sila savlada silu opruge koja pritišče iglu, igla se podiže i počinje ubrizgavanje. Podizaj igle može se vidjeti na slici 26. kao blaga udubina u dijagramu struje koja nastaje zbog gibanja metala u magnetnom polju. Struja zatim raste do konačne vrijednosti definirane naponom akumulatora i otporom brizgaljke. Kod zatvaranja brizgaljke, kada se strujni krug otvori, dolazi do naglog porasta napona koji se ograničava diodom (D1, slika 25.) ili aktivnom hvataljkom. Do porasta napona dolazi zbog protivljenja zavojnice promjeni struje. Da bi se tok struje održao, u zavojnici se inducira napon koji ovisi o brzini pada struje. Kako se tranzistorom struja može prekinuti gotovo trenutno, inducirani napon može dosegnuti stotine volti što bi dovelo do uništenja samog tranzistora. Zbog toga se napon ograničava na oko 50V čime se osigurava dovoljno brzi pad struje, a time i brzo zatvaranje brizgaljke bez štetnih posljedica za tranzistor. Za upravljanje brizgaljkama odabran je integrirani krug MC33810 od NXP-a. MC33810 sadrži četiri kanala za upravljanje brizgaljkama i četiri kanala za upravljanje IGBT-ovima koji se koriste za prekidanje struje prema bobinama. Pomoću SPI komunikacije omogućuje slanje grešaka kao što su otvoreni krug, kratki spoj i pregrijavanje *divera* te automatsko gašenje pojedinog *drivera* u slučaju kratkog spoja. Sve to nalazi se u jednom integriranom krugu pakiranja SOIC32 okvirnih dimenzija 10x10mm zbog čega se znatno smanjuje broj komponenata i površina zauzeta na tiskanoj pločici.



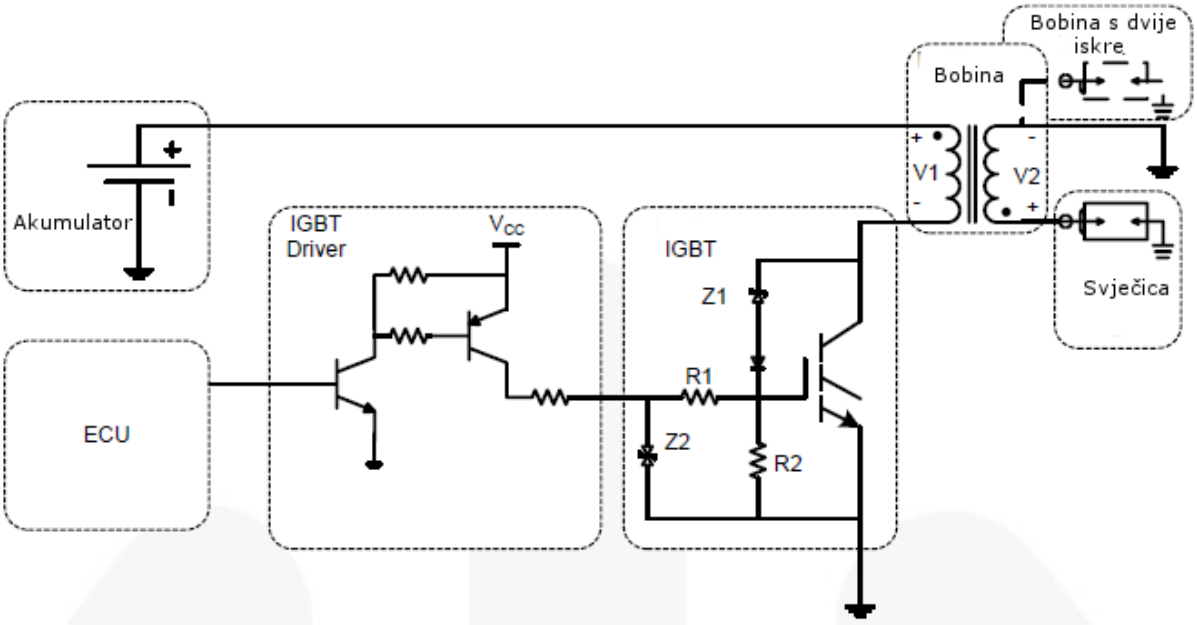
Slika 27. Integrirani krug MC33810 za upravljanje brizgaljkama i bobinama

Brizgaljke se spajaju izravno na izlaze OUT0-OUT3 na integriranom krugu, a princip rada sličan je prikazanom na slici 25. Pripadajući ulazi DIN0-DIN3 povezani su s mikrokontrolerom. Ako je napon na ulazu DINx manji od $0.2 \cdot V_{DD,}$ MOSFET na izlazu OUTx ne provodi i brizgaljka je zatvorena. Ako je napon na ulazu iznad $0.7 \cdot V_{DD}$, MOSFET provodi i brizgaljka se otvara. V_{DD} je napon na kojem se nalaze logički krugovi unutar MC33810, a može biti 3.3V ili 5V. Izlazi GDx koriste se kao *driveri* za IGBT bobina, a biti će objašnjeni u sljedećem dijelu zajedno sa sustavom paljenja.

Postoji nekoliko različitih strategija zaštite *drivera* brizgaljki od kratkog spoja koje je moguće programirati preko SPI-a. Kod prve strategije kratkog spoja ograničava se struja kroz *driver*, a u slučaju da kratki spoj potraje dulje od 60μs *driver* se potpuno gasi. Druga strategija uključuje praćenje temperature *drivera* te se za vrijeme kratkog spoja ograničava struja dok temperatura *drivera* ne prijeđe dozvoljenu vrijednost. U tom trenutku *driver* se privremeno gasi dok temperatura ne padne ispod vrijednosti histereze. Treća strategija je kombinacija prve dvije, ovisno da li se prije dostigne granična temperatura ili trajanje od 60μs. Otvoreni krug detektira se mjerenjem napona na odvodu MOSFETA. Kada je MOSFET u isključenom stanju, na odvodu postoji blagi *pull-down* prema masi. Kod ispravnog kruga napon na odvodu trebao bi biti oko 12V tj. jednak naponu akumulatora. U slučaju otvorenog kruga odvod više nije povezan preko brizgaljke na 12V već samo preko *pull-downa* na masu. Otvoreni krug detektira se kada napon na odvodu padne ispod granične vrijednosti. Za slučaj kada je MOSFET u uključenom stanju, otvoreni krug detektira se mjerenjem struje. Ako je struja manja od 200mA strujni krug se smatra otvorenim.

3.3.2. Sustav paljenja

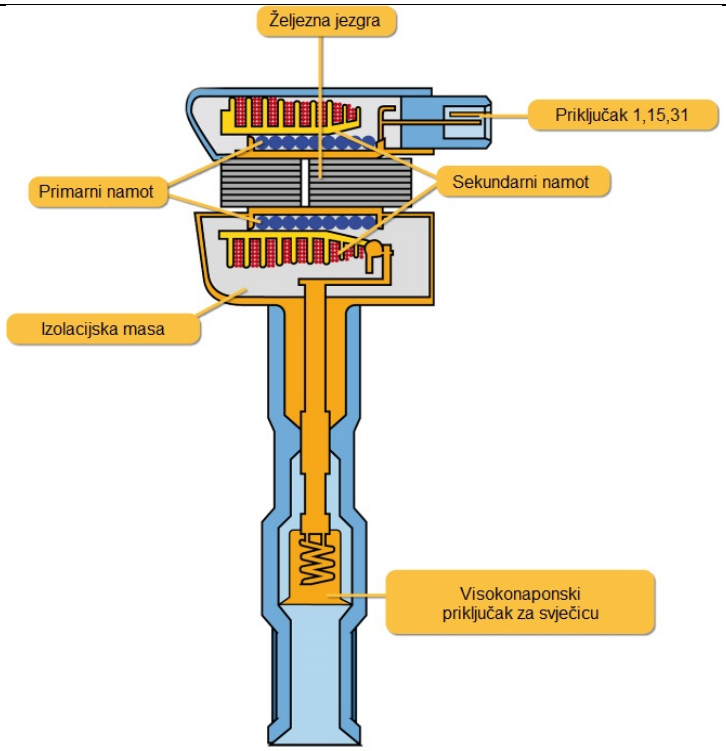
Danas najčešće korišten tip paljenja je induktivno paljenje čiji se rad temelji na principu Faradayevog zakona indukcije. Visoki napon dobiva se naglom promjenom struje, a time i magnetnog polja zavojnice. Suvremeni induktivni sustav paljenja čine: bobina, IGBT (*Insulated-gate bipolar transistor*) s pripadajućim *driverom*, svjećica i upravljačka jedinica(ECU).



Slika 28. Shematski prikaz suvremenog sustava paljenja[7]

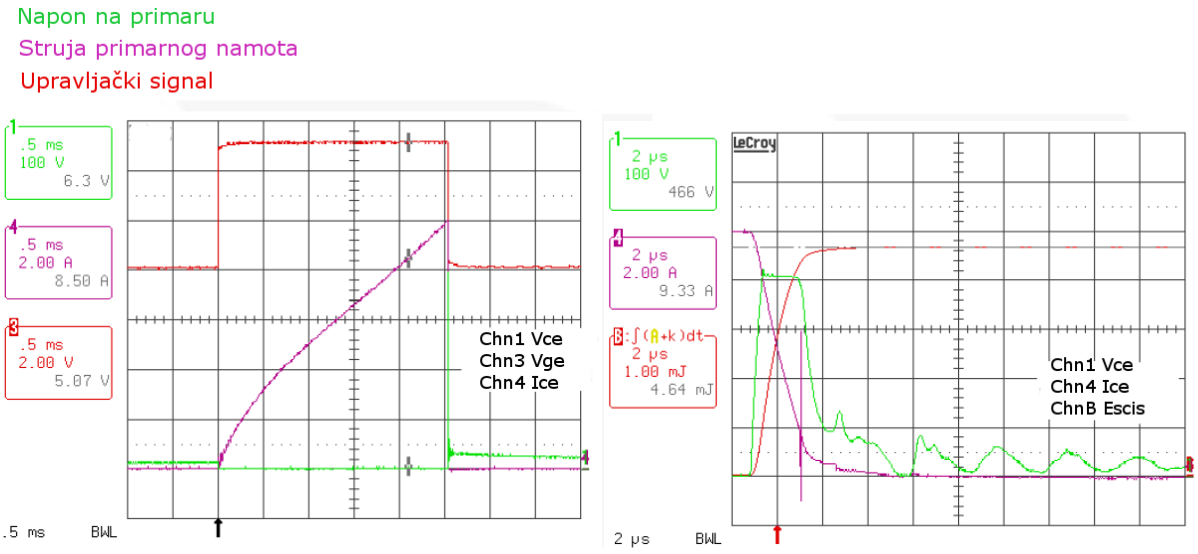
Upravljačka jedinica na temelju podataka o temperaturi, opterećenju,brzini vrtnje i nizu drugih parametara šalje signal *driveru* IGBT-a koji pali ili gasi IGBT i tako propušta i prekida struju kroz primarnu zavojnicu u bobini.

Bobina je u osnovi transformator. Jezgru čini paket lamela oko kojeg je namotan sekundarni visokonaponski namot iz tanke bakrene žice izolirane lakom. Primarni namot napravljen je iz debele bakrene žice čiji je jedan kraj spojen na priključak 15 (+), a drugi povezan sa krajem sekundarnog namota i spojen na priključak 1 (kolektor IGBT-a). Suvremene bobine imaju integrirani IGBT tako da je za njihovo aktuiranje potreban samo upravljački signal od 5V.



Slika 29. Presjek suvremene bobine

Centralna jedinica daje signal (crvena linija, slika 30.) i uključuje IGBT koji počinje voditi struju. Brzina porasta struje kroz primarni namot (roza linija) određena je induktivitetom zavojnice i njezinim otporom. Kako je sekundarni namot otvoreni krug (zazor na svjećici), energija se privremeno pohranjuje u magnetnom polju zavojnice.



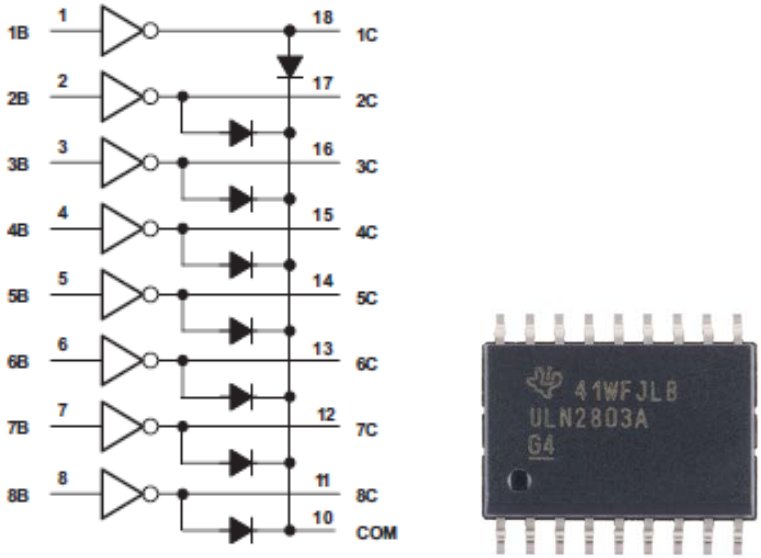
Slika 30. Punjenje i pražnjenje bobine(lijevo), pražnjenje uvećano(desno)[7]

Kada upravljački signal padne na 0V, IGBT se isključuje i dolazi do naglog pada struje u primarnom namotu. Brza promjena struje uzrokuje naglu promjenu magnetskog polja zbog čega se inducira visoki napon u primaru. Ovaj visoki napon ograničen je hvataljkom u IGBT-u na 300V-600V (zelena linija) zbog zaštite samog IGBT-a i izolacije zavojnice. U sekundarnom namotu inducira se napon proporcionalan omjeru namota primara i sekundara. Suvremeni IGBT moduli imaju automatsko ograničavanje struje kako bi se spriječilo pregrijavanje bobine i samog modula te *soft shutdown* funkciju koja u slučaju predugog vremena punjenja polako smanjuje struju kroz primarni namot da ne dođe do preskakanja iskre na svjećici.

MC33810 koristi se kao *pre-driver* za IGBT. Moguće ga je podesiti na dva različita moda rada: *Ignition (IGBT) gate pre-driver* i *General purpose gate pre-driver*. Prvi mod rada posjeduje određene funkcije specifične za upravljanje sustavima paljenja, a koristi informacije sa ulaza FBx kako bi mogao detektirati greške kratkog spoja, otvorenog kruga, otvorenog sekundarnog kruga, ograničavanje struje kroz primarnu zavojnicu i *soft shutdown*. Budući da je kod većine suvremenih bobina IGBT integriran u njezino kućište, nije moguće koristiti povratnu informaciju na ulazu FBx jer se fizički nije moguće spojiti na kolektor IGBT-a. Međutim to ne predstavlja problem zato što bobine s integriranim modulima već posjeduju navedene funkcije. Iz tog razloga koristi se *General purpose gate pre-driver* mod bez povratnih informacija na FBx ulazima. Mjerenje struje kroz primarnu zavojnicu moguće je izvesti pomoću otpornika na ulazima RSP i RSN (Slika 27.). Otpornik R=20mΩ postavlja se u seriju s primarnom zavojnicom bobine, a pad napona mjeri se na navedenim ulazima. Poznavanjem pada napona i otpora moguće je izračunati struju. Informacija o struji dobiva se na NOMI i MAXI izlazima, ako je struja manja od vrijednosti programirane za nominalnu i maksimalnu struju izlaz će biti logički 0, u suprotnom logički 1. Ako struja prijeđe maksimalnu definiranu vrijednost *driver* se automatski gasi.

3.3.3. Izlazi opće namjene

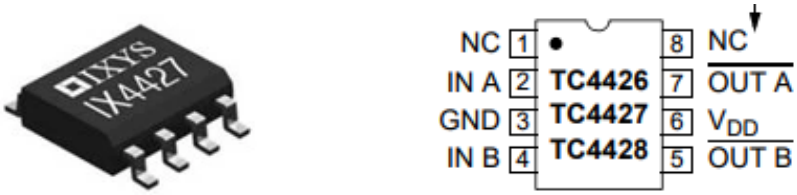
U ovu skupinu pripadaju izlazi za upravljanje relejima pumpe goriva i ventilatora te ostalih uređaja koji se mogu upravljati putem releja ili izravno, ovisno o jakosti struje. Za upravljanje relejima koristi se za to namijenjen integrirani krug ULN2803A koji sadrži 8 Darlington tranzistorskih parova s zaštitnim diodama.



Slika 31. Integrirani krug ULN2803 s osam bipolarnih tranzistora

1B-8B su ulazi na baze bipolarnih tranzistora koji se spajaju izravno na mikrokontroler. Izlazi 1C-8C vode na kolektor tranzistora preko kojega se zavojnica releja spaja na masu.

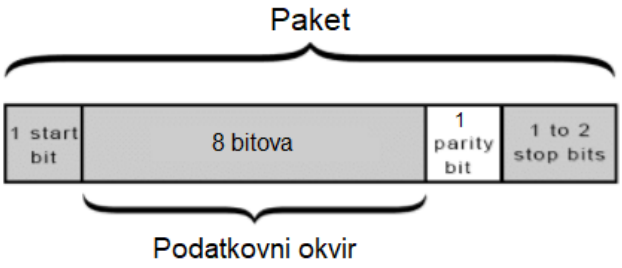
Uređaji koji zahtijevaju bržu aktuaciju, kao što su elektromagnetski ventili pneumatskog sustava za promjenu stupnja prijenosa, upravljaju se tranzistorima. Za tu svrhu odabran je dvokanalni N-MOSFET STS8DN6LF6AG. *Gate driver* IX4427 koristi se za upravljanje MOSFET-om kako bi se mikrokontroler zaštitio u slučaju pojave tranzijenata i postiglo brže paljenje i gašenje MOSFET-a.



Slika 32. Integrirani krug IX4427 za punjenje upravljačke elektrode MOSFET-a

3.3.4. Komunikacija s računalom

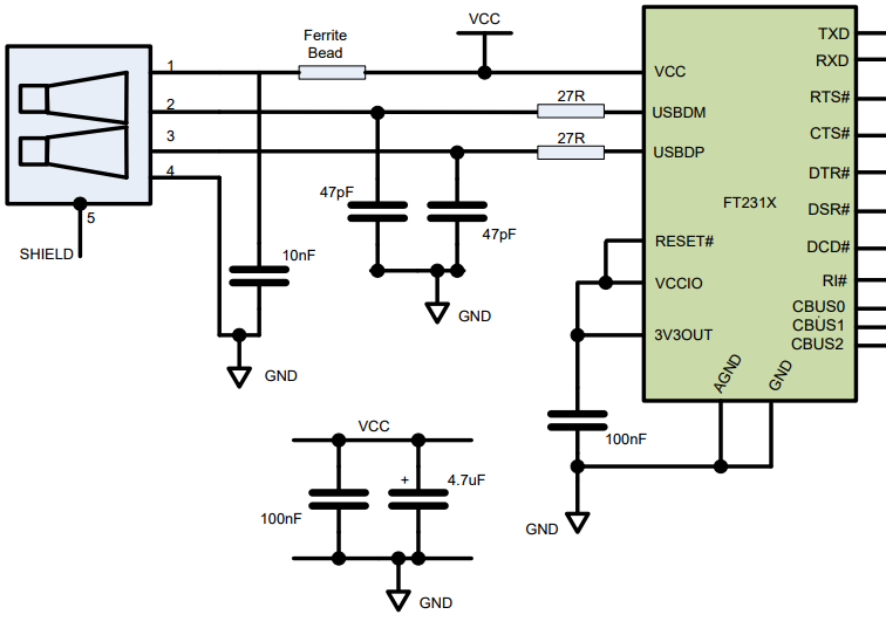
Komunikacija s računalom potrebna je zbog podešavanja i nadgledanja parametara rada motora u realnom vremenu. Za tu svrhu odabrana je asinkrona serijska komunikacija zbog relativno jednostavne implementacije u softwareu i zadovoljavajuće brzine. UART (Universal Asynchronous Receiver/Transmitter) je uređaj integriran unutar mikrokontrolera koji pretvara paralelnu komunikaciju iz CPU-a u serijsku. Za komunikaciju su potrebne dvije žice, jedna za slanje i jedna za primanje podataka (Rx i Tx). UART šalje podatke asinkrono što znači da nema zasebnog signala koji će sinkronizirati primatelja i pošiljatelja već se početak i kraj poruke identificira start i stop bitovima. Kada UART primatelj detektira start bit, počinje čitati dolazeće bitove sa određenom frekvencijom koja se naziva *baud rate*. Za uspješnu komunikaciju oba UART-a moraju biti podešena na isti *baud rate* i istu strukturu paketa. Podatci su organizirani u pakete, a svaki paket sadrži start bit, 8 bitova podataka, paritet i stop bit.



Slika 33. Paket podataka kod serijske komunikacije

UART Tx linija nalazi se na logički 1 dok nema prijenosa podataka. Početak slanja označava se spuštanjem Tx linije na logički 0 za jedan ciklus. Kada UART primatelj detektira prijelaz sa logički 1 na 0 počinje sa čitanjem bitova u podatkovnom okviru definiranom frekvencijom. Paritet se koristi za provjeru ispravnosti poslanih podataka tj. jesu li se bitovi u podatkovnom okviru promijenili tijekom slanja zbog različitog *baud ratea*, elektromagnetskih smetnji i slično. Nakon što UART primatelj pročita svih 8 bitova unutar podatkovnog okvira, pobroji koliko bitova ima vrijednost 1 i provjeri da li je zbroj paran ili neparan broj. Ako je paritet 0 (parni), broj bitova sa vrijednosti 1 mora biti paran broj. Ako je paritet 1 (neparni), broj bitova sa vrijednosti 1 mora biti neparan broj. Kada se paritet poklapa s podatcima, UART smatra da je prijenos podataka izvršen bez grešaka. Paket završava tako da se Tx linija digne na logički 1 u trajanju barem 2 ciklusa.

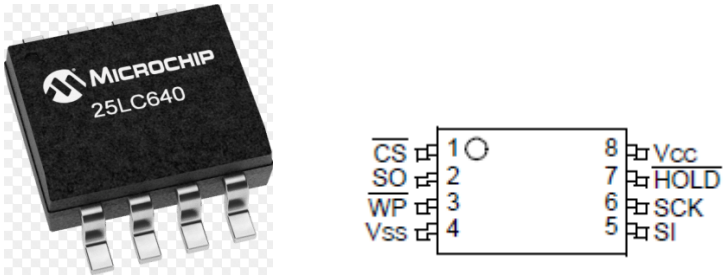
Za komunikaciju s računalom potreban je UART-USB konverter koji rješava sve vezano uz konverziju UART-a na USB protokol, a sam uređaj (ECU) se na računalu prikazuje kao Virtualni Com Port(VPC). Integrirani krug koji se koristi za tu svrhu je FT231X prikazan na slici 34. zajedno sa preporučenom shemom spajanja. Uređaj se s mikrokontrolrom povezuje preko TXD i RXD priključaka (Rx na mikrokontroleru spaja se sa TXD na FT231X i obratno), a priključci USBDM i USBDP spajaju se na odgovarajuće žice na USB kabelu.



Slika 34. Shema spajanja UART-USB konvertera

3.3.5. Memorija za trajno pohranjivanje podataka

EEPROM (Electricaly Erasable Programmable Read-Only Memory) je električno izbrisiva programabilna ispisna memorija za trajno pohranjivanje manjih količina podataka. Imaju ograničen broj pisanja i brisanja podataka koji se kod modernih EEPROM-a kreće oko 10^6 ciklusa. Danas se najčešće programiraju serijskim putem (SPI, I2C, 1-wire, Microwire) jer se na taj način dobivaju manji uređaji s manjim brojem pinova. EEPROM koji se koristi u ECU-u je 25LC640A od Microchipa, memorije 64kbit programiran preko SPI-a. U njemu se pohranjuju kalibracije, *look up* tablice i sve postavke promjenjive od strane korisnika.



Slika 35. Memorija za trajno pohranjivanje podataka-EEPORM 25LC640

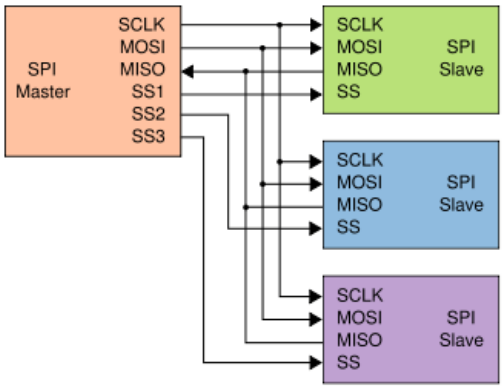
Slika 35. prikazuje EEPROM i funkcije pojedinih pinova. Vcc (3.3V) i Vss (GND) su napajanje uređaja, WP zaštita od pisanja, a SO, SI, SCK i CS signali SPI komunikacijskog protokola. SPI komunikacijski protokol koristi se za brzo i pouzdano slanje podataka na manje udaljenosti. Za komunikaciju su potrebne četiri linije:

MOSI (SI) (Master Out Slave In) – kada mikrokontroler radi kao *master*, ovo je linija za slanje podataka, a kada radi kao *slave* za primanje podataka

MISO (SO) (Master In Slave Out) – Kada mikrokontorler radi kao *master*, ovo je linija za primanje podataka, a kada radi kao *slave* za slanje podataka

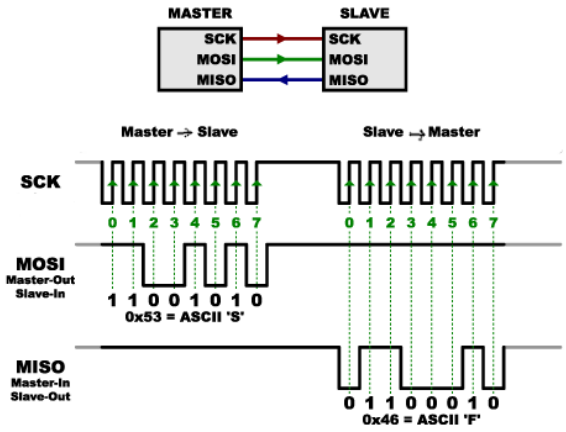
SCK (SPI Clock) – linija za takt pod kojim se izvodi komunikacija, takt daje *master*

CS (Chip Select) – ovom linijom upravlja master, a koristi se za odabiranje trenutno aktivnog *slave*-a. Kada je CS na visokoj razini, slave ignorira podatke koji dolaze iz MOSI linije. Kada se CS dovede na nisku razinu, komunikacija sa *slave*-om započinje.



Slika 36. Povezivanje više uređaja SPI komunikacijskim protokolom

U svakom trenutku smije biti samo jedan *master*. Ako se koristi više *slave*-ova, svaka CS linija povezuje se na drugi pin na *master*-u kako bi se omogućilo biranje *slave*-a s kojim se trenutno komunicira (Slika 36.). Spuštanjem CS linije na nisku razinu i zapisivanjem bajta u *data* registar započinje prijenos podataka MOSI linijom.

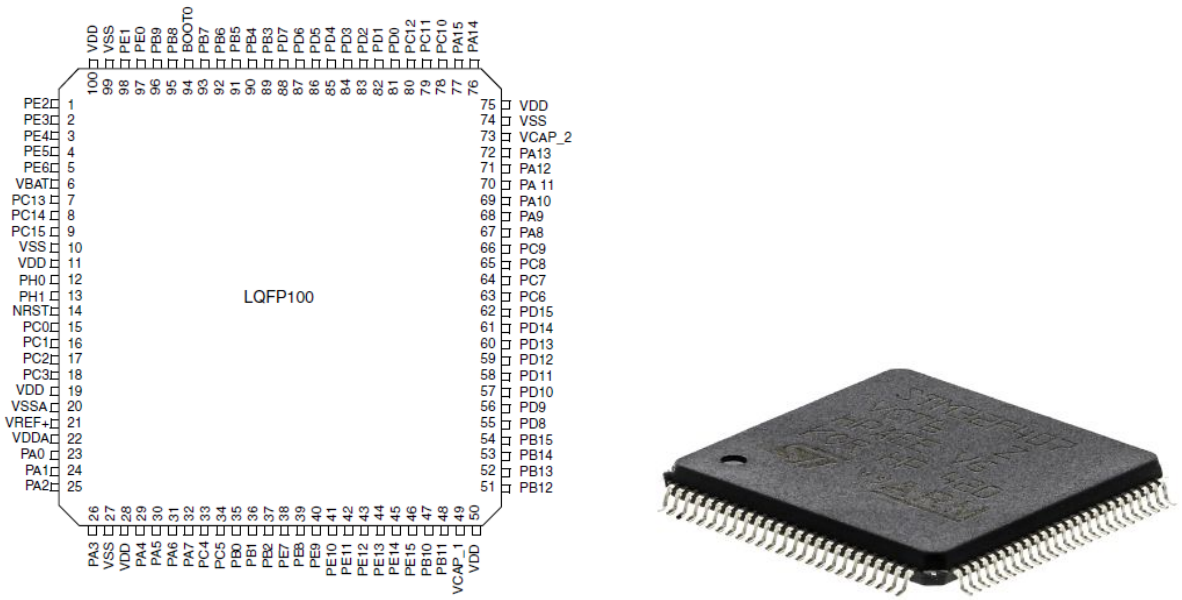


Slika 37. Signali na linijama kod SPI komunikacije

Bitovi se šalju redom jedan iza drugoga, ako se šalje bit čija je vrijednost 0 onda MOSI linija za vrijeme uzlaznog signala takta SCK mora biti na niskoj razini i obrnuto (Slika 37). Nakon završetka slanja *slave* odgovara slanjem bajta *masteru*. Zapisivanje podataka u EEPROM biti će objašnjeno kasnije zajedno sa upravljačkim programom.

3.4. Mikrokontroler

Mikrokontroler je centralni dio upravljačke jedinice motora. U njemu je pohranjen program koji sadrži sve algoritme i podatke potrebne za upravljanje radom motora na temelju podataka dobivenih iz senzora. Odabrani mikrokontroler je STM32F407VGT6 od proizvođača STMicroelectronics, temeljen na ARM Cortex-M4 32-bitnoj jezgri s radnom frekvencijom 168MHz, 1Mbyte flash memorije i 192Kbyte SRAM. Sadrži tri 12-bitna analgno digitalna konevrtera(ukupno 16 ulaza), 2 digitalno analogna konvertera, dvanaest 16-bitnih brojača i dva 32-bitna brojača. Od komunikacijskih modula sadrži tri I2C, tri SPI-a, šest UART-a, CAN, SDIO i USB. Dozvoljena radna temperatura je od -40 do 105°C, a radni napon od 1.8-3.6V.



Slika 38. Mikrokontroler STM32F407VGT6 proizvođača STMicroelectronics

Ovaj mikrokontroler odabran je zbog zadovoljavajuće brzine i funkcija koje posjeduje, ali i lake dobavljivosti i niske cijene razvojne pločice koja je korištena pri testiranju prototipa. Jedan od razloga odabira STM32 serije mikrokontrolera je i postojanje STM32duino jezgre i biblioteka koje pojednostavljuju podešavanje samog mikrokontrolera i omogućuju programiranje bez direktnog pristupanja registrima. Kučište mikrokontrolera je 100-pinsko (LQFP100) s 80 ulazno-izlaznih vodova. Svi ulazno-izlaznih vodovi mogu se koristiti kao općeniti digitalni ulazi ili izlazi dok su neki od njih povezani s perifernim uređajima i posjeduju posebne funkcije kao što su ADC, brojači, UART, SPI i ostalo.

Ostali vodovi koriste se za napajanje mikrokontrolera, povezivanje na vanjski kristal koji daje takt, podešavanje boot moda i reset. U tablici 4. prikazane su funkcije dostupne na pojedinim vodovima koje će se koristiti u ECU-u. Portove je moguće podesiti za različite alternativene funkcije, a ove prikazane u tablici odnose se na prve po važnosti.

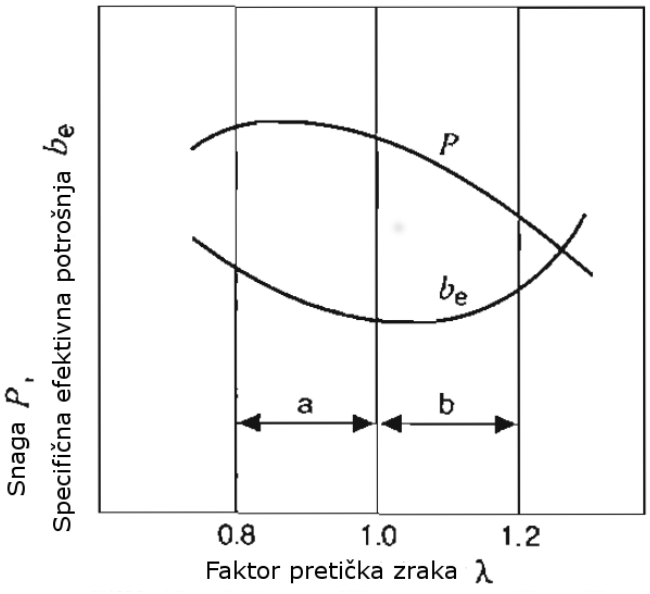
Tablica 4. Osnovne funkcije nekih pinova STM32F407 mikrokontrolera

PERIFERNI UREĐAJI		**PIN**
ADC	ADC1	PA0-PA7
	ADC2	PB0-PB1
	ADC3	PC0-PC5
Brojači	TIM1	PA8-PA11
	TIM2	PA0-PA3
	TIM3	PA6, PA7, PB0, PB1
	TIM4	PB6-PB9
UART	UART1	PA9, PA10
	UART2	PA2, PA3
	UART3	PB10, PB11
SPI	SPI	PB12-PB15

4. Upravljanje Ottovim motorom

4.1. Priprema smjese

Ottovi motori u osnovi zahtijevaju upravljanje s dva glavna parametra, a to su početak preskakanja iskre i količina ubrizganog goriva. Početak ubrizgavanja kod Ottovih motora nije ključan za rad, ali može utjecati na potrošnju goriva i emisiju ispušnih plinova. Motori s ubrizgavanjem goriva u usisnu cijev rade s homogenom smjesom, a snaga se regulira isključivo promjenom masenog protoka u cilindar pomoću zaklopke u usisnoj cijevi. Homogena smjesa upaljiva je u uskom području faktora pretička zraka λ=0.6...1.6 te je zbog toga radno područje ovih motora ograničeno na $0.8<\lambda<1.2$. Stehiometrijski omjer zraka i goriva za benzin iznosi 14.7:1 što znači da je za potpuno izgaranje 1kg benzina teoretski potrebno 14.7kg zraka. U realnim uvjetima stehiometrijskim omjerom goriva i zraka ne postiže se niti najmanja potrošnja goriva niti najveća snaga. Rad sa stehiometrijskom smjesom koristi se isključivo zbog ispravnog funkcioniranja trokomponentnog katalizatora cestovnih vozila koja moraju zadovoljiti stroge zahtjeve emisija ispušnih plinova. S obzirom da natjecateljski automobili ne moraju zadovoljavati takve zahtjeve, faktor pretička zraka može se podesiti za najveću snagu ili minimalnu potrošnju goriva.



Slika 39. Utjecaj faktora pretička zraka na snagu i specifičnu efektivnu potrošnju goriva[1]

Najveća snaga postiže se kod lagano bogate smjese (λ=0.85...0.9). Kako je snaga najčešće ograničena količinom zraka koju je moguće dovesti u cilindar bitno je osigurati da što više raspoloživih molekula kisika reagira sa gorivom, što se postiže nešto bogatijom smjesom od stehiometrijeske. Najmanja specifična efektivna potrošnja goriva postiže se kod nešto siromašnije smjese (λ=1.05...1.1), a višak zraka omogućuje da što više molekula goriva reagira s kisikom. Povećavanjem faktora pretička zraka naglo opada brzina izgaranja, a granična vrijednost kod koje se izgaranje još uvijek može ispravno odvijati ovisi o samoj konstrukciji motora i sustava usisa te se najčešće kreće oko λ=1.2...1.4.

U stacionarnim uvjetima rada motora (konstantna brzina vrtnje i opterećenje) količina ubrizganog goriva tj. duljina otvorenosti brizgaljke određuje se iz glavne tablice ubrizgavanja prema trenutnoj brzini vrtnje i opterećenju. Osim stacionarnih uvjeta rada pri normalnoj radnoj temperaturi, postoji još niz drugih režima koji zahtijevaju određene korekcije u količini ubrizganog goriva, a to su: hladan start, faza nakon pokretanja, faza zagrijavanja, tranzijente faze kod otvaranja i zatvaranja zaklopke i kočenje motorom.

Hladan start

Kod hladnog starta dolazi do povećane kondenzacije goriva na hladne stjenke cilindra i usisnih kanala te slabijeg isparavanja i miješanja goriva i zraka. Ako se količina ubrizganog goriva ne poveća, smjesa će biti presiromašna te će doći do izostanka paljenja. Za kompenzaciju ovih pojava tijekom hladnog starta potrebno je ubrizgati dodatnu količinu goriva koja će nadomjestiti slabo isparavanje i kondenzaciju.

Faza nakon pokretanja

Kratko vrijeme nakon pokretanja potrebno je ubrizgavati dodatnu količinu goriva dok se cilindar i glava ne zagriju i poboljšaju unutarnje formiranje smjese.

Faza zagrijavanja

Ova faza nastupa nekoliko sekundi nakon pokretanja motora. Još uvijek postoji potreba za dodatnom količinom goriva zbog kondenzacije na usisnim kanalima i slabijeg isparavanja. Dodatna količina ubrizganog goriva progresivno se smanjuje porastom temperature rashladne tekućine sve do postizanja radne temperature.

Tranzijentne faze

Tlak u usisnoj grani ima veliki utjecaj na isparavanje goriva. Pri nižem tlaku u usisnoj grani gorivo lakše isparava pa je i količina goriva koja se kondenzira na stjenke manja. Nagle promjene tlaka do kojih dolazi zbog otvaranja ili zatvaranja zaklopke uzrokuju promjene u

količini goriva kondenziranog na stjenkama. Kod naglog porasta tlaka sloj goriva na stjenkama se povećava pa se dio ubrizganog goriva troši na kondenzaciju. Zbog toga je potrebno ubrizgavati dodatnu količinu goriva dok se ne uspostavi ravnoteža u graničnom sloju. Količina goriva koju je potrebno ubrizgati osim o brzini promjene tlaka ovisi i o temperaturi stijenki.

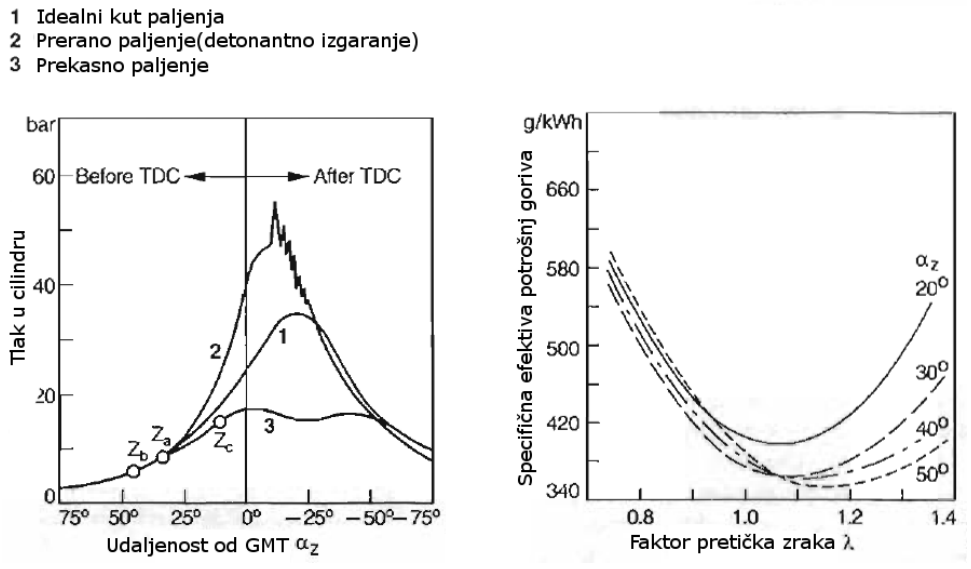
Kočenje motorom

Kod kočenja motorom ubrizgavanje goriva može se u potpunosti prekinuti jer se od motora ne zahtjeva da proizvodi moment. Time se može dobiti mala ušteda u potrošnji goriva.

4.2. Paljenje smjese

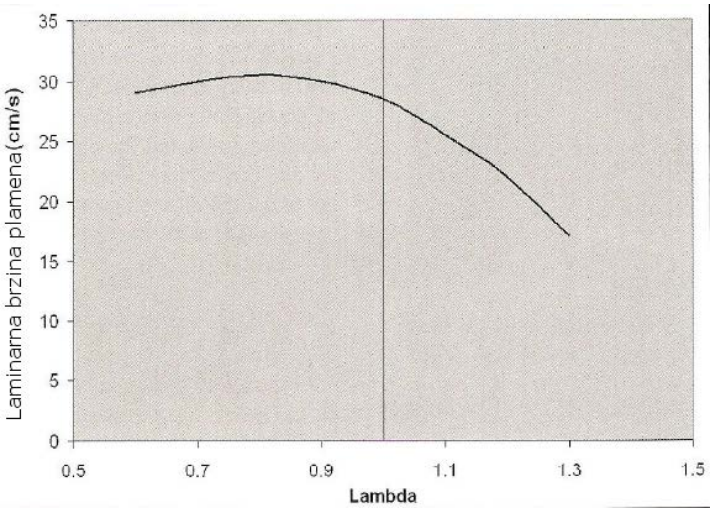
Funkcija sustava paljena je da inicira izgaranje komprimirane smjese goriva i zraka u točno određenom trenutku. U Ottovom motoru to se postiže preskakanjem iskre na svjećici. Energija iskre potrebna za sigurno upaljivanje ovisi o omjeru goriva i zraka. Kod stehiometrijske smjese to iznosi oko 0.2mJ, a kod siromašne i bogate smjese do 3mJ. Energija je u stvarnosti znatno veća kako bi se u svim uvjetima omogućilo sigurno paljenje smjese. Osim energije bitan parametar je i električni napon na elektrodama. Napon potreban za preskakanje iskre ovisi o tlaku u cilindru u trenutku paljenja i iznosi oko 10kV.

Dva osnovna parametra kojima je potrebno upravljati kod sustava paljenja su vrijeme punjenja primarne zavojnice bobine i trenutak preskakanja iskre. Vrijeme punjenja ovisi o induktivitetu i otporu primarne zavojnice i naponu akumulatora. Kako su otpor i induktivitet zavojnice konstantni, potrebno vrijeme punjenja mijenja se samo zbog promjene napona.



Slika 40. Utjecaj kuta pretpaljenja na tlak u cilindru i specifičnu efektivnu potrošnju goriva[1]

Trenutak preskakanja iskre definira se kao kut u stupnjevima zakreta koljenastog vratila prije ili poslije gornje mrtve točke. Kut paljenja izravno utječe na snagu motora, potrošnju goriva i pojavu detonantnog izgaranja. Slika 40. lijevo prikazuje utjecaj kuta pretpaljenja na tlak u cilindru. Premali kut pretpaljenja ima za posljedicu produljeno izgaranje, manju snagu motora, povećane temperature ispušnih plinova i time pregrijavanje motora dok preveliki kut pretpaljenja uzrokuje veliki porast tlaka i temperature u blizini GMT, pregrijavanje i sklonost detonantnom izgaranju. Optimalni kut paljenja ovisi o nekoliko parametara od kojih su najvažniji: brzina vrtnje, opterećenje, temperatura, omjer goriva i zraka, korišteno gorivo i oblik prostora izgaranja. Gorivo i oblik prostora izgaranja su konstantni pa je kut paljenja potrebno podešavati u ovisnosti o preostala četiri parametra. Zakašnjenje paljenja je približno vremenski konstantno za isti omjer goriva i zraka i kompresijski omjer, zbog čega se kod veće brzine vrtnje proteže preko većeg kuta zakreta koljenastog vratila. Zato se povećanjem brzine vrtnje paljenje pomiče na ranije. S druge strane, povećanjem brzine vrtnje raste brzina strujanja i vrtloženje u cilindru što dovodi do bržeg širenja fronte plamena i djelomičnog poništavanja efekta zbog zakašnjenja paljenja. Povećanjem opterećenja (stupnja punjenja) mijenjaju se temperatura i tlak na kraju kompresije čime se smanjuje zakašnjenje paljenja i raste brzina izgaranja zbog čega je kut paljenja potrebno pomaknuti na kasnije (bliže GMT). Temperatura rashladne tekućine i zraka utječu na sličan način. Kod hladnog motora zakašnjenje paljenja je nešto veće pa se kut pretpaljenja povećava.



Slika 41. Laminarna brzina plamena u ovisnosti o faktoru pretička zraka[12]

Bogatstvo smjese također ima značajan utjecaj na zakašnjenje paljenja i brzinu izgaranja. Slika 41. prikazuje ovisnost laminarne brzine širenja plamena o faktoru pretička zraka. Najveća brzina postiže se kod faktora pretička zraka λ=0.85...0.9, a opada za bogatije i siromašnije smjese pa je sukladno tome potrebno korigirati i kut pretpaljenja.

4.3. Upravljački program

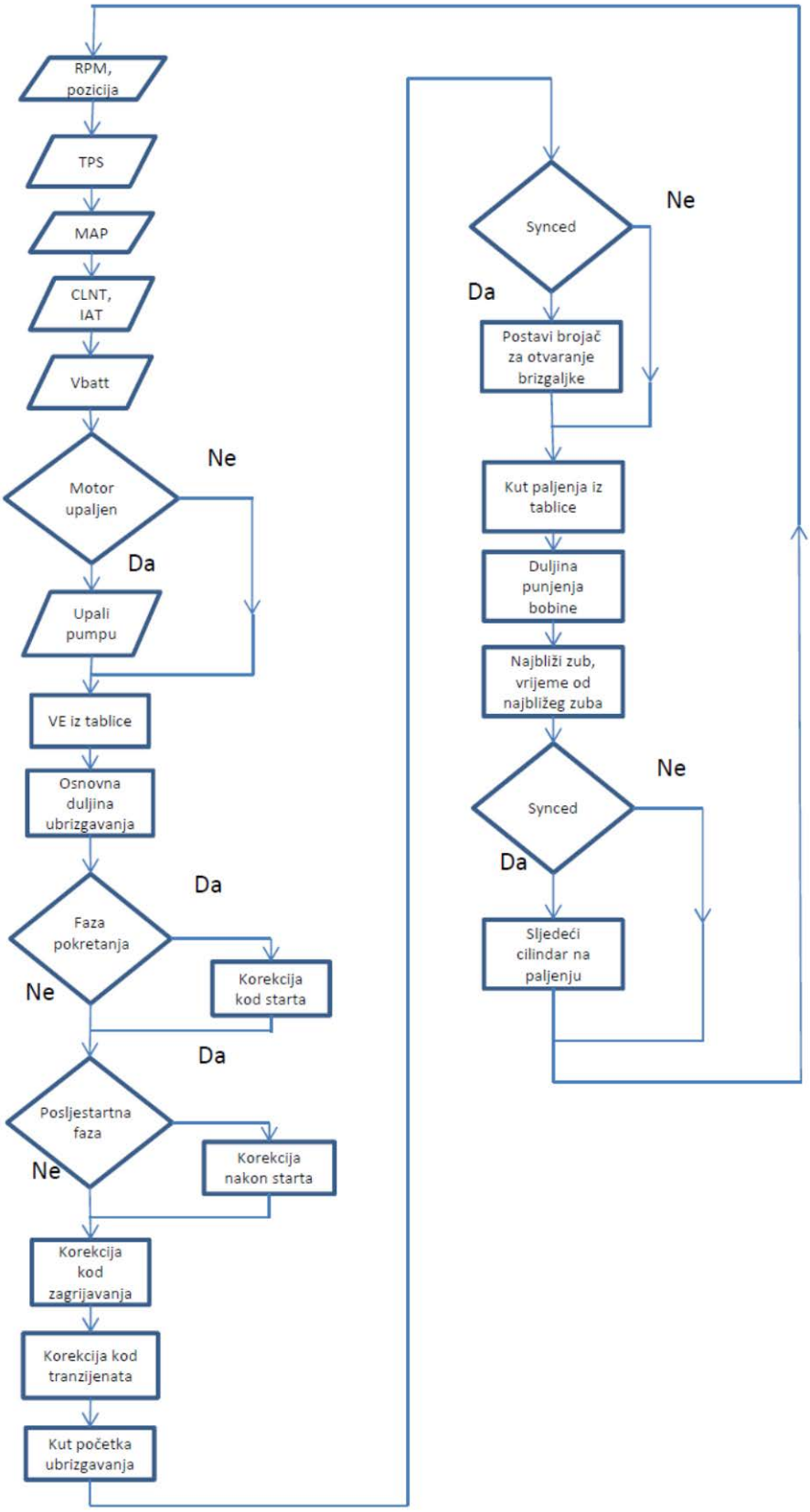
Upravljački program mikrokontrolera napisan je u C++ programskom jeziku uz korištenje GNU GCC kompajlera. Visual Studio sa priključkom Visual Micro i STM32duino knjižnicama korišten je kao razvojna okolina. Upravljački program pohranjen je u flash memoriji mikrokontrolera, a programiran je pomoću odgovarajućeg programatora STLinkV2. Program je zadužen za upravljanje trenutkom preskakanja iskre i duljinom otvorenosti brizgaljke prema unaprijed definiranim mapama i korekcijskim faktorima te upravljanje popratnim uređajima kao što su ventilatori, pumpa goriva i slično.

Program se izvodi u beskonačnoj petlji. Očitavanje vrijednosti sa senzora i računanje trajanja ubrizgavanja i kuta pretpaljenja te ostali vremenski manje kritični zadatci izvode se u glavnoj petlji dok se očitavanje senzora koljenastog vratila, otvaranje i zatvaranje brizgaljki i paljenje i gašenje bobine izvodi u prekidnim funkcijama (*interrupts*). Mikrokontroler s računalom komunicira pomoću UART-USB modula gdje se povezuje s aplikacijom koja omogućuje nadziranje i podešavanje svih potrebnih parametara rada motora u realnom vremenu. Osnovni dijagram toka upravljačkog programa prikazan je na slici 42.

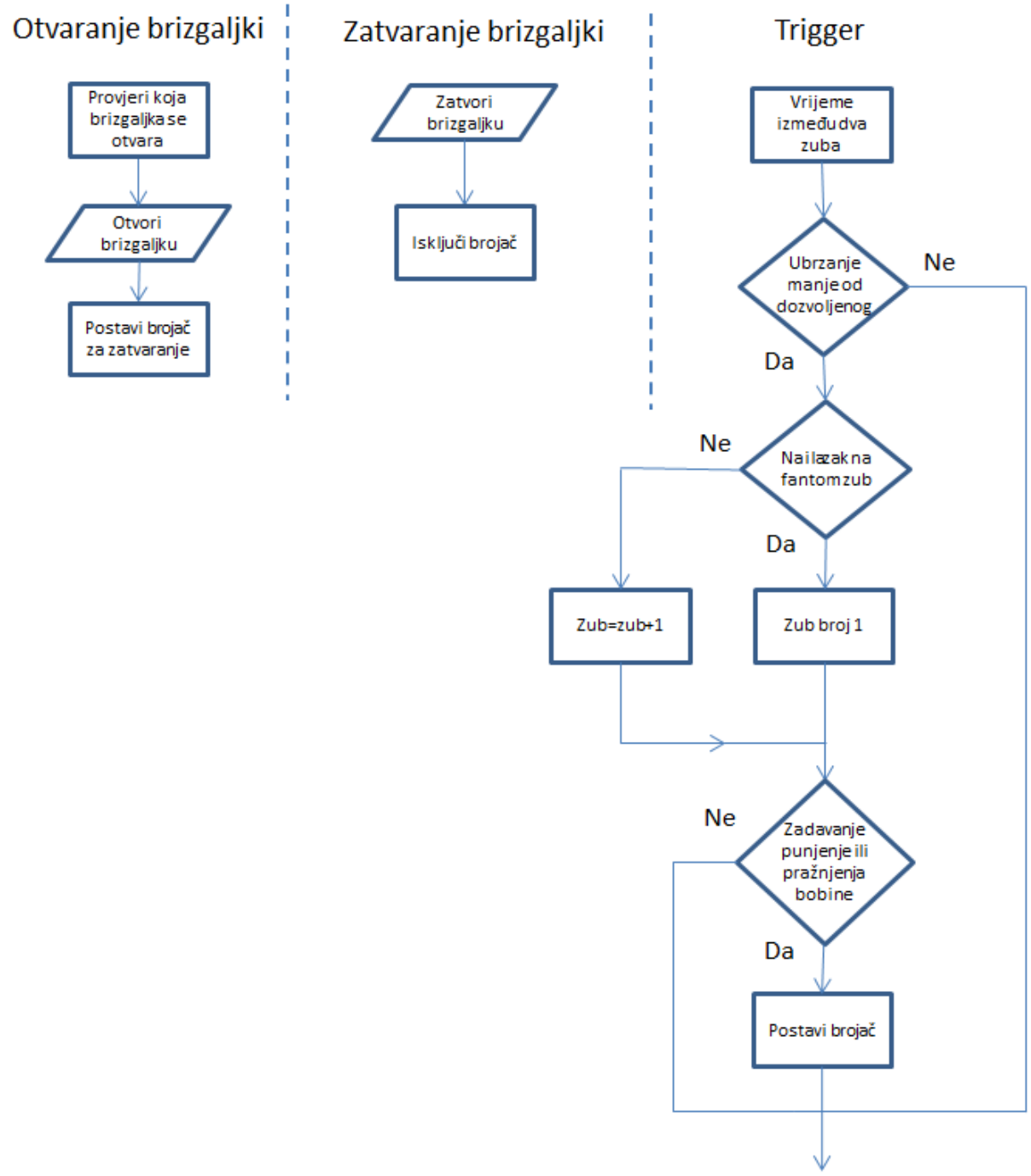
Nakon pokretanja programa, prije ulaska u glavnu petlju, podešavaju se ulazni i izlazni kanali, brojači, vanjske prekidne funkcije i komunikacije. Svi digitalni ulazni kanali postavljaju se kao *input floating* (nisu spojeni niti na GND niti Vcc), a analogni ulazi kao *anaalog input*. Izlazni kanali podešeni su kao *push pull* tj. spojeni su na Vcc ili na GND, za razliku od open collectora koji može biti GND ili *floating*. Vanjske prekidne funkcije postavljaju se na ulaze čije je signale potrebno trenutno registrirati, kao što je senzor zakreta koljenastog vratila. Ako signal promjeni vrijednost s visokog na niski napon (1 na 0) ili obratno, ovisno o postavkama, generira se prekid koji zaustavlja izvođenje glavne petlje i skače na dio programa u prekidnoj funkciji. Od komunikacija se koriste dva UART-a, jedan za komunikaciju s računalom, a drugi za komunikaciju sa *datalogger-om* te SPI za komunikaciju s driverima brizgaljki i EEPROM-om.

Čitanje senzora koljenastog vratila vremenski je kritično jer o njemu ovisi određivanje pozicije koljenastog vratila koje je ključno za određivanje trenutka preskakanja iskre. Zato se na njegovom ulazu koristi *external interrupt* funkcija tj. vanjski prekid, koja kod promjene vrijednosti signala trenutno zaustavlja izvođenje glavne petlje i skače na izvođenje prekidne funkcije. Unutar funkcije mjeri se vremenski razmak između dva zuba na enkoderu i prema njemu računa brzina vrtnje. Ako je razlika prethodnog i trenutnog vremenskog razmaka prevelika tj. fizikalno nemoguća, prekid se zanemaruje kao da do njega nije niti došlo. Ako je vrijeme između dva zuba veće od prethodnog za neki faktor (ovisno o broju zuba koji nedostaje), prepoznaje se praznina, a zub nakon praznine označava se kao prvi zub. U programu je podešen zub koji označava GMT i prema njemu se izračunava na kojem zubu treba zadati punjenje ili pražnjenje bobine. Kada se naiđe na taj zub (zub koji je najbliže poziciji punjenja ili pražnjenja bobine), postavlja se brojač koji mora generirati prekid za vrijeme potrebno da se koljenasto vratilo zakrene za kut jednak razlici trenutnog kuta i kuta punjenja ili pražnjenja bobine. Kako je točna pozicija poznata samo u trenutku prolaska preko zuba, sve pozicije između određuju se pomoću vremena koje je potrebno za neki kutni pomak pri trenutnoj brzini vrtnje.

U glavnoj petlji najprije se očitavaju analogne vrijednosti sa senzora koje se zatim pretvaraju u stvarne fizikalne veličine. Ako je motor pokrenut tj.brzina vrtnje je veća od 0, pali se pumpa goriva. U slučaju da brzina vrtnje naraste iznad dozvoljene, onemogućuje se punjenje bobine i na taj način sprječava preskakanje iskre. Zatim se određuje volumetrijska efikasnost iz tablice prema trenutnoj radnoj točki i računa osnovna duljina ubrizgavanja. Na osnovnu duljinu ubrizgavanja nadodaju se korekcije koje ovise o trenutnom režimu rada i temperaturi. Iz *lookup* tablice određuje se kut početka ubrizgavanja i podešava se brojač koji generira prekid u trenutku otvaranja brizgaljke. U prekidnoj funkciji podešava se drugi brojač koji će zatvoriti brizgaljku za izračunato vrijeme. Kut paljenja također se određuje iz *lookup* tablice prema trenutnoj radnoj točki, a duljina punjenja bobine prema naponu baterije. Prema kutu paljenja izračunava se najbliži zub na kojem će se u prekidnoj funkciji (*trigger*) postaviti brojač za punjenje ili pražnjenje bobine i vrijeme za koje će se generirati prekid. U ostatku petlje provjeravaju se uvjeti za upravljanje perifernim uređajima i obavlja se komunikacija s računalom.



Slika 42. Pojednostavljeni dijagram toka glavne petlje upravljačkog programa



Slika 43. Dijagram toka prekidnih funkcija

Očitavanje vrijednosti senzora osnovna je stvar za računanje svih upravljačkih parametara. Senzori se očitavaju u glavnoj petlji programa koja se izvodi oko 4000 puta u sekundi. Očitavanje analognih ulaza radi se pozivanjem funkcije *analogRead*() koja pokreće AD konverziju i daje trenutnu vrijednost napona na odabranom pinu. STM32 posjeduje analogno digitalne pretvarače s 12bitnom razlučivosti što znači da vrijednost napona kod očitavanja

može poprimiti 2^{12} različitih vrijednosti tj. 0...4095 za iznose napona 0...3.3V. Nakon očitavanja iz ADC registra, vrijednost je korisno pretvoriti u veličinu koja ima fizikalno značenje kako bi se olakšalo daljnje računanje.

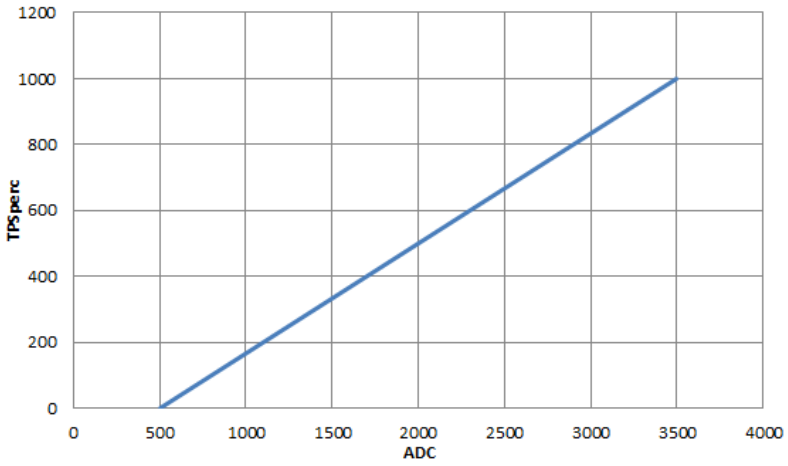
4.3.1. Očitavanje položaja zaklopke

Senzor položaja zaklopke je linearni potenciometar kojemu je izlazni napon linearno ovisan o kutu zakreta. Položaj zaklopke stoga se iz očitane vrijednosti ADC-a izračunava jednadžbom pravca. Poznavanjem vrijednosti napona za potpuno zatvorenu i otvorenu zaklopku moguće je dobiti jednadžbu pravca iz koje se zatim izračunava položaj za sve ostale vrijednosti napona:

$$TPSperc = \frac{TPS\,\max - TPS\,\min}{ADC\,\max - ADC\,\min} \cdot ADC - \frac{(TPS\,\max - TPS\,\min) \cdot ADC\,\min}{ADC\,\max - ADC\,\min} + TPS\,\min \qquad \textbf{(14)}$$

gdje su:

- *TPS*perc – trenutna pozicija zaklopke
- *TPS*max – otvorena zaklopka (100%)
- *TPS*min – zatvorena zaklopka (0%)
- *ADC*max – ADC vrijednost za TPSmax
- *ADC*min – ADC vrijednost za TPSmin
- *ADC* – trenutna vrijednost ADC-a



Slika 44. Primjer karakteristike senzora pozicije zaklopke

Slika 44. prikazuje primjer ovisnosti pozicije zaklopke *TPS*perc i izmjerenog napona *ADC* za nasumično odabrane vrijednosti *ADC*min=500 i *ADC*max=3500. *TPS*max postavljen je na 1000 zbog veće preciznosti očitavanja. Težilo se da što više matematičkih operacija bude s tipom varijable *integer* tj. cjelobrojnim varijablama jer je njihovo izvođenje znatno brže nego *floating point* operacije. Zato se kod računanja uzima da *TPSperc* poprima vrijednosti od 0 do 1000 čime se dobiva rezolucija od 0.1% bez korištenja *floating point* aritmetike.

4.3.2. Očitavanje tlaka u usisnom kolektoru

Izračunavanje tlaka iz izlaznog napona MAP senzora temelji se na istom principu kao i kod senzora pozicije zaklopke. Ovisnost izlaznog napona i tlaka MAP senzora je linearna pa je dovoljno poznavati napone za dva različita tlaka. Kalibracija je nešto kompliciranija nego kod zaklopke jer je potrebno mijenjati tlak i mjeriti ga manometrom ili nekim drugim kalibriranim senzorom.

$$MAP = \frac{MAP_2 - MAP_1}{ADC_2 - ADC_1} \cdot ADC - \frac{(MAP_2 - MAP_1) \cdot ADC_1}{ADC_2 - ADC_1} + MAP_1 \qquad \textbf{(15)}$$

gdje su:

MAP – trenutni tlak

MAP_2 – mjereni tlak kod kalibracije

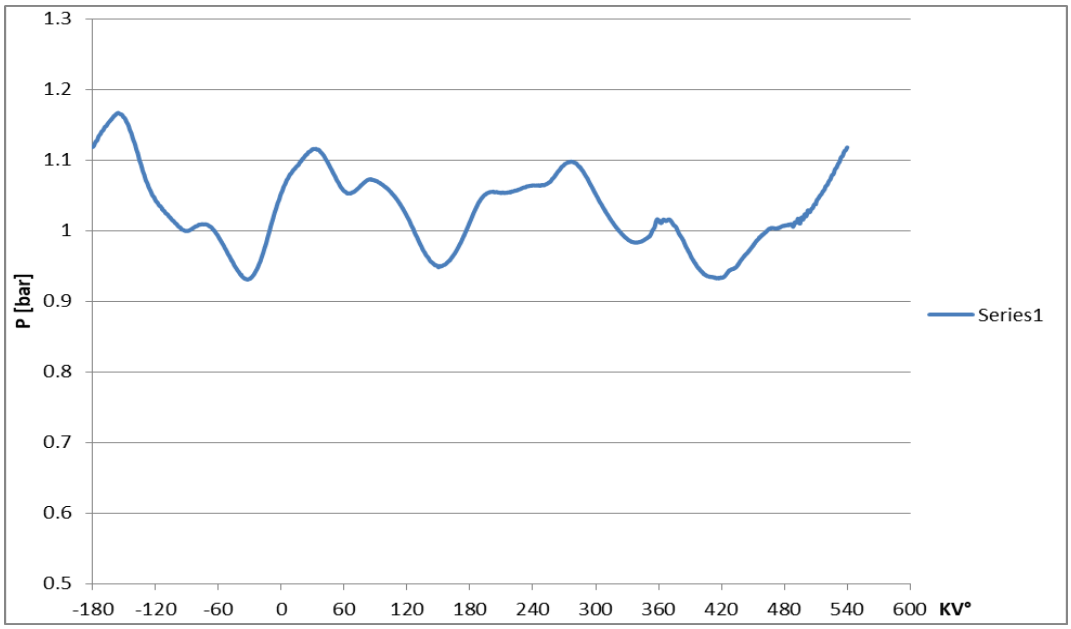
MAP_1 – mjereni tlak kod kalibracije

ADC_2 – ADC vrijednost za MAP_2

ADC_1 – ADC vrijednost za MAP_1

ADC – trenutna vrijednost ADC-a

Zbog velikih varijacija tlaka u usisnom kolektoru tijekom jednog ciklusa, pogotovo kod jednocilindričnih motora, očitavanje vrijednosti tlaka u nasumičnim trenutcima može dovesti do variranja količine ubrizganog goriva i kuta pretpaljenja u istoj radnoj točki. Za primjer se može uzeti motor pri punom opterećenju (zaklopka potpuno otvorena) i konstantnoj brzini vrtnje. Dijagram na slici 45. prikazuje tlak u usisnoj cijevi kroz jedan ciklus za takav slučaj.



Slika 45. Primjer profila tlaka u usisu jednocilindričnog motora

Izmjereni tlak značajno ovisi o kutu zakreta koljenastog vratila na kojemu se uzima uzorak pa je bitno uvijek uzorkovati pri istom položaju koljenastog vratila za taj cilindar. Kod jednocilindričnih motora to znači uzorkovanje jednom u ciklusu, a kod višecilindarskih motora onoliko puta koliko ima cilindara. Kod četverocilindarskih motora profil tlaka se u usisnom kolektoru ponavlja četiri puta u ciklusu pa se i mjerenje može izvršiti ukupno četiri puta za istu poziciju pojedinog cilindra. Kut zakreta na kojem se uzima uzorak podesiv je u programu, a kod svakog mjerenja uzima se nekoliko uzoraka za redom i izračunava srednja vrijednost. Točan kut uzimanja uzorka nije kritičan, samo će glavna mapa ubrizgavanja imati nešto drugačije vrijednosti za postizanje istog faktora pretička zraka. Bitno je da se nakon podešavanja mape ubrizgavanja i paljenja trenutak uzorkovanja ne mijenja jer se mogu dobiti drugačije vrijednosti tlaka za istu radnu točku. Na ovaj način dobiva se stabilan tlak od ciklusa do ciklusa.

4.3.3. Senzori temperature zraka i rashladne tekućine

Senzor temperature rashladne tekućine ključan je za ispravan rad motora kod hladnog starta i u fazi zagrijavanja jer se prema temperaturi izračunavaju sve korekcije kojima se množi duljina otvorenosti brizgaljke dobivena iz glavne mape ubrizgavanja. Temperatura zraka također utječe na potrebnu količinu ubrizganog goriva jer se promjenom temperature mijenja

gustoća zraka, a time i masa zraka koja ulazi u cilindar. Oba senzora rade na principu mijenjanja otpora promjenom temperature. Senzor sa dodatnim otporom tvori naponsko dijelilo (Slika 14.) na čijem se izlazu promjenom temperature mijenja napon. Iznos otpora vanjskog otpornika je R1=2.2kΩ, a izlazni napon jednak je:

$$V_{out} = V_{cc} \frac{R_2}{R_1 + R_2} = 5 \cdot \frac{R_2}{2200 + R_2} \quad \textbf{(16)}$$

Svi senzori napajaju se s 5V, dok je najveći dozvoljeni napon na analognim ulazima mikrokontrolera 3.3V zbog čega na ulazima postoji dodatno naponsko dijelilo koje snižava napon za faktor 0.6. Napon na ulazu mikrokontrolera jednak je:

$$V_{mcu} = 5 \cdot \frac{R_2}{2200 + R_2} \cdot 0.6 = 3 \cdot \frac{R_2}{2200 + R_2} \quad \textbf{(17)}$$

Odgovarajuća vrijednost ADC-a za neki napon *V* izračunava se iz omjera:

$$\frac{V}{3.3} = \frac{ADC}{4095} \Rightarrow ADC = V \frac{4095}{3.3} \quad \textbf{(18)}$$

Iz čega slijedi ovisnost otpora senzora i *ADC*-a:

$$ADC == 3 \cdot \frac{4095}{3.3} \cdot \frac{R_2}{2200 + R_2} \quad \textbf{(19)}$$

Temperatura se izračunava tako da se najprije očita napon, a zatim iz *lookup* tablice temperatura koja odgovara tom naponu. Vrijednosti temperatura između onih definiranih u tablici 5. određuju se linearnom interpolacijom. *Lookup* tablica pohranjena je kao jednodimenzijski niz sa 14 elemenata,a svaki element odgovara određenom naponu. Razlika napona između bilo koja dva susjedna elementa jednaka je da bi se za određivanje rednog broja elementa mogla koristiti jednadžba pravca.

Tablica 5. Ovisnost temperature o otporu i naponu senzora

Element u nizu	Temperatura	ADC	R	Vout
0	-40	3600	64533.3	4.84
1	-25	3330	18654.2	4.47
2	-11	3060	10158.0	4.11
3	-2	2790	6580.7	3.75
4	6	2520	4609.5	3.38
5	13	2250	3361.1	3.02
6	20	1980	2499.5	2.66
7	28	1710	1869.1	2.30
8	36	1440	1387.8	1.93
9	45	1170	1008.3	1.57
10	55	900	701.4	1.21
11	69	630	448.1	0.85
12	90	360	235.5	0.48
13	130	90	54.5	0.12

Nakon što se očita vrijednost *ADC*-a, jednadžbom pravca dobiva se redni broj elementa u nizu za taj napon. Ako se uzme da je *y* element niza, a *x* vrijednost *ADC*-a jednadžba pravca glasi:

$$y = \frac{y_2 - y_1}{x_2 - x_1} \cdot x - \frac{(y_2 - y_1) \cdot x_1}{x_2 - x_1} + y_1 = \frac{13 - 0}{90 - 3600} \cdot x - \frac{(13 - 0) \cdot 3600}{90 - 3600} + 0 \quad \textbf{(20)}$$

$$y = -0.0037\,\mathrm{x} + 13.333 \quad \textbf{(21)}$$

Za provjeru se može uzeti npr. *ADC*=2655 :

$$y = -0.0037 \cdot 2655 + 13.333 = 3.5 \quad \textbf{(22)}$$

Dobivena vrijednost nalazi između trećeg i četvrtog člana niza što je točno. Težinskim prosjekom vrijednosti trećeg i četvrtog člana izračunava se temperatura:

$$T = (y - y_L) \cdot \mathrm{T}\left[y_H\right] + (y_H - y) \cdot \mathrm{T}\left[y_L\right] \quad \textbf{(23)}$$

Gdje su:

y_L – član zaokružen na nižu vrijednost

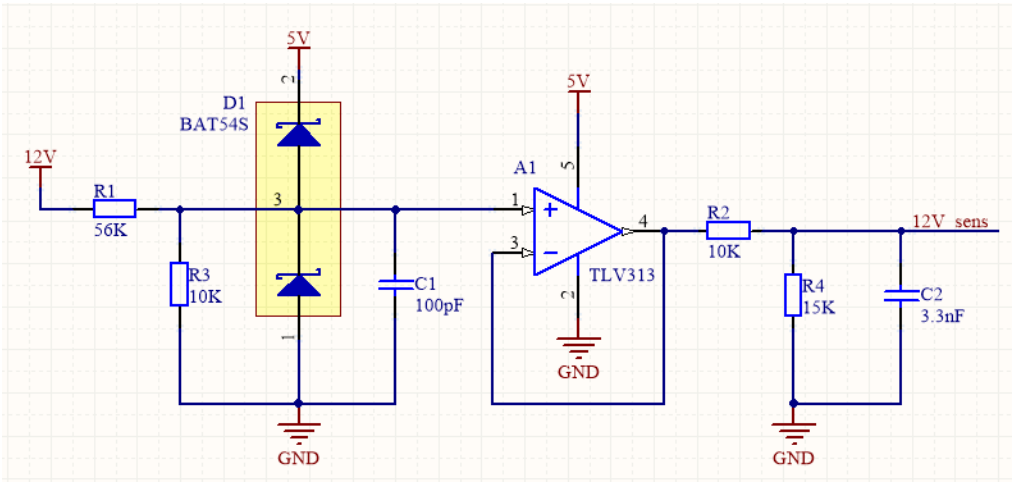
y_H – član zaokružen na višu vrijednost

$T[\ ]$ – temperatura za odgovarajući član niza

Otpor i napon koji odgovaraju određenim elementima niza su fiksni i prikazani u tablici 2. Kalibracija se izvodi unošenjem temperature za unaprijed definirane otpore, a vrijednosti u tablici 5. odgovaraju Bosch NTC senzoru (0 280 130 026) koji se trenutno koristi na bolidu Strix.

4.3.4. Mjerenje napona akumulatora

Napon baterije ključan je parametar za određivanje mrtvog vremena brizgaljke koje se nadodaje na izračunato trajanje ubrizgavanja te za određivanje trajanja punjenja bobine.



Slika 46. Zaštitni krug analognog ulaza za mjerenje napona

Napon sa akumulatora prije ulaska u mikrokontroler prolazi kroz zaštitni krug na slici 46. što uključuje dva naponska dijelila, jedan prije i jedan nakon operacijskog pojačala. Konačni napon na analognom ulazu izračunava se prema izrazu:

$$V_{in} = V_{batt} \cdot \frac{10}{56+10} \cdot \frac{15}{10+15} = \frac{V_{batt}}{11} \qquad \textbf{(24)}$$

Očitavanjem analognog ulaza dobiva se vrijednost *ADC*-a koja se zatim preračunava u ulazni napon,a množenjem ulaznog napona sa faktorom naponskog dijelila izračunava se napon baterije.

$$V_{batt} = 11 \cdot V_{in} = 11 \cdot \frac{ADC}{4095} \cdot 3.3 \quad \textbf{(25)}$$

Napon V_{batt} pohranjen je u varijabli tipa *integer* pa se gornja jednadžba množi sa 100 čime se eliminiraju decimalna mjesta uz zadržavanje rezolucije od 0.01V.

4.3.5. Mjerenje faktora pretička zraka

Faktor pretička zraka mjeri se uskopojasnom ili širokopojasnom lambda sondom. Uskopojasna lambda sonda generira napon od 0-1V koji se može izravno mjeriti mikrokontrolerom. Zbog svoje izlazne karakteristike, vrlo je precizna oko λ=1, ali je neupotrebljiva u području bogate i siromašne smjese. Širokopojasna lambda sonda s druge strane zahtjeva složeni kontroler koji upravlja grijačem i transportnim člankom, ali je zato upotrebljiva u širokom rasponu faktora pretička zraka 0.65<λ<zrak. Izlazni signal iz korištenog kontrolera (SLC OEM) je napon 0-5V koji linearno ovisi o faktoru pretička zraka λ=0.68...1.36. Signal prolazi kroz zaštitni krug s naponskim djelilom koje spušta napon za faktor 0.6. Najprije se iz vrijednosti ADC-a izračuna napon:

$$V_{\lambda} = \frac{ADC}{4095} \cdot 3.3 \quad \textbf{(26)}$$

Zatim se jednadžbom pravca kroz dvije točke dobije ovisnost faktora pretička zraka o vrijednosti *ADC*-a:

$$\lambda = 0.01827 \cdot ADC + 68 \quad \textbf{(27)}$$

4.3.6. Izračunavanje duljine otvorenosti brizgaljke

Duljina otvorenosti brizgaljke glavni je parametar kojim se utječe na količinu ubrizganog goriva. Za određivanje potrebne količine goriva, a zatim i duljine otvorenosti brizgaljke potrebno je što preciznije odrediti masu zraka koja je ušla u cilindar. Poznavanjem mase zraka u cilindru i karakteristike brizgaljke moguće je okvirno odrediti duljinu ubrizgavanja koja se zatim množi nizom korekcijskih faktora određenih u postupku kalibracije s ciljem postizanja željenog faktora pretička zraka.

Najčešće tri korištene metode za određivanja masenog protoka zraka ovisno o senzorima na kojima se temelje su:

- Izravnim mjerenjem pomoću senzora masenog protoka zraka
- Mjerenjem apsolutnog tlaka u usisnom kolektoru(*Speed density*)
- Mjerenjem otvorenosti zaklopke(*Alpha-N*)
- Kombinacije navedenih metoda

Senzorom masenog protoka zraka može se postići najveća preciznost određivanja masenog protoka jer je ovo jedina metoda kojom se protok mjeri, a ne procjenjuje. Ovo je najčešća metoda korištena na serijskim vozilima kod kojih se zahtijeva održavanje faktora pretička zraka u vrlo uskom području zbog kontrole emisija ispušnih plinova. Na bolidu Strix se ne koristi jer je kompliciraniji za implementaciju nego MAP senzor, osjetljiv je na nečistoće, predstavlja restrikciju u usisnoj cijevi i osjetljiv je na puštanje zraka nakon zaklopke u slučaju lošeg brtvljenja spojeva.

Speed density metodom protok se ne mjeri izravno nego se procjenjuje na temelju tlaka u usisnom kolektoru, temperaturi zraka, brzini vrtnje i unaprijed definiranoj tablici stupnja punjenja. Poznavanjem tlaka, temperature zraka, volumena cilindra i plinske konstante može se jednadžbom idealnog plina izračunati masa zraka u cilindru koja se zatim množi sa stupnjem punjenja.

Alpha-N metoda koristi senzor pozicije zaklopke kao glavni parametar za određivanje potrebne količine goriva. Glavna mapa prema kojoj se određuje količina ubrizganog goriva koristi poziciju zaklopke i brzinu vrtnje. Maseni protok ne može se pretpostaviti bez poznavanja točne karakteristike zaklopke pa je podešavanje manje intuitivno nego kod *speed density* modela. Problem ovog modela je nemogućnost prepoznavanja promjene tlaka zbog

npr. promjene visine i teško podešavanje kod niskih brzina vrtnje zbog velike promjene opterećenja za malu promjenu kuta zaklopke. Koristan je kod motora s jednom zaklopkom po cilindru kod kojih tlak može varirati toliko da bude neupotrebljiv za određivanje opterećenja. Na bolidu se trenutno koristi *speed density* metoda određivanja masenog protoka koja će biti opisana u nastavku.

Kao što je prije rečeno, masa zraka u cilindru kod *speed density* metode određuje se pomoću jednadžbe idealnog plina. Potrebno je izračunati masu zraka koja bi stala u cilindar pri standardnim uvjetima i zatim duljinu otvorenosti brizgaljke tako da se dobije stehiometrijska smjesa. Izračunata duljina otvorenosti brizgaljke je konstantna jer ovisi o volumenu cilindra, protoku brizgaljke, stehiometrijskom omjeru goriva i zraka, standardnom tlaku, temperaturi i plinskoj konstanti koji se ne mijenjaju tijekom rada motora. Masa zraka koja stane u cilindar pri standardnim uvjetima jednaka je:

$$m_z = \frac{pV_{cyl}}{RT} = \frac{101325 \cdot V_{cyl}}{286.9 \cdot 293.15}[kg] \quad \textbf{(28)}$$

Gdje je:

- p=101325Pa - tlak pri standardnim uvjetima
- R=286.9 J/molK - individualna plinska konstanta za zrak
- T=293.15K - temperatura pri standardnim uvjetima
- V_{cyl} - volumen cilindra [m^3]

Masa goriva dobiva se dijeljenjem mase zraka sa stehiometrijskim omjerom zraka i goriva:

$$m_g = \frac{m_z}{Z}[\text{kg}] \quad \textbf{(29)}$$

Gdje je:

- Z=14.7 stehiometrijski omjer zraka i goriva za benzin ili Z=9.8 za E85

Poznavanjem masenog protoka brizgaljke može se izračunati duljina ubrizgavanja za traženu masu goriva. Maseni protok najčešće je deklariran u g/min i potrebno ga je pretvoriti u kg/ms:

$$\dot{m}_g = \dot{m}_g \cdot \frac{1}{60 \cdot 1000 \cdot 1000} \left[\text{kg/ms}\right] \tag{30}$$

Slijedi duljina ubrizgavanja u milisekundama za traženu masu goriva:

$$reqPW = \frac{m_g}{\dot{m}_g} \left[\text{ms}\right] \tag{31}$$

Ovako dobivena duljina ubrizgavanja zatim se množi s omjerom mjerenog i standardnog tlaka te volumetrijskom efikasnosti i lambdom iz *lookup* tablice što su ujedno i glavni parametar za podešavanje duljine ubrizgavanja:

$$PW = reqPW \cdot \frac{VE}{100} \cdot \frac{MAP}{101} \cdot \frac{1}{lambda} + injDead \tag{32}$$

- *PW* – zadana duljina ubrizgavanja
- *VE* - volumetrijska efikasnost
- *MAP* - apsolutni tlak u usisnom kolektoru
- *injDead* – mrtvo vrijeme brizgaljke
- *lambda* – zadani faktor pretička zraka, pohranjen u tablici kao i volumetrijska efikasnost

S obzirom da je ovisnost količine ubrizganog goriva i duljine otvorenosti brizgaljke linearna(osim kod jako kratkih duljina otvorenosti), onda će *VE* i *lambda* linearno utjecati na faktor pretička zraka. Volumetrijska efikasnost predstavlja postotak mase zraka koja uđe u cilindar u odnosu na masu izračunatu jednadžbom idealnog plina pri trenutnom tlaku u usisnom kolektoru. Osim o brzini vrtnje i tlaku u usisnom kolektoru, volumetrijska efikasnost

ovisi o profilu bregastog vratila i konfiguraciji kompletnog sustava usisa i ispuha. Kod bilo kakvih modifikacija motora koje dovode do promjene stupnja punjenja cilindra potrebno je napraviti korekcije u tablici da bi se održao željeni faktor pretička zraka. Volumetrijska efikasnost pohranjena je u dvodimenzijskom nizu f1[18][18] (prvi broj odnosi se na retke, a drugi na stupce) varijable tipa *byte* u ovisnosti o trenutnom tlaku i brzini vrtnje(Slika 47.).

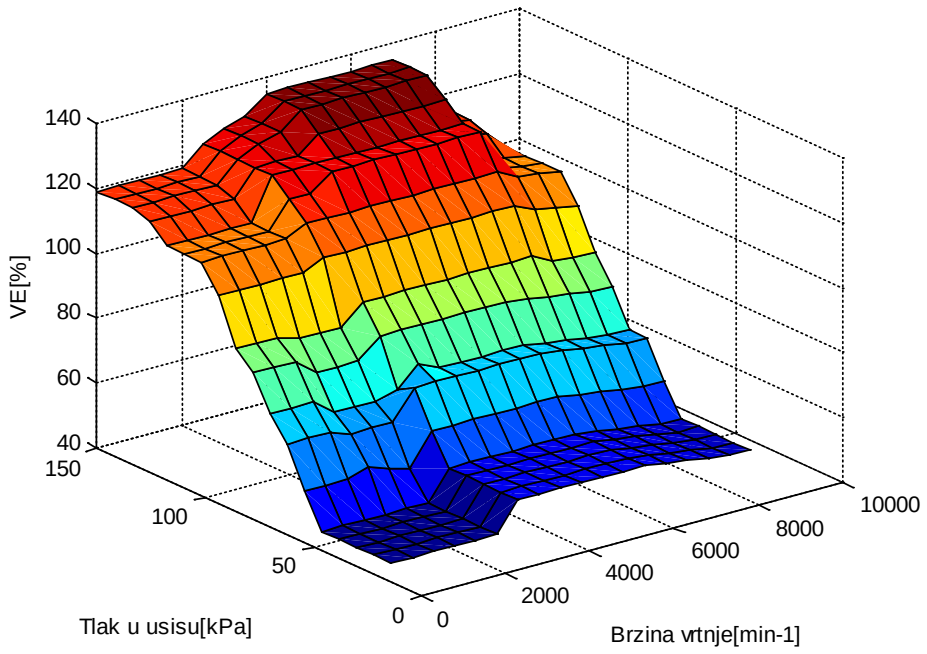
```
byte f1[18][18] =      {

{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,  110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,  110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,  110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,  110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,  110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,  110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,  110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  125,  125,  130,  130,  130,  125,  120,  110,  110,  110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  125,  125,  125,  125,  125,  125,  120,  115,  110,  110,  110,  110 },
{ 120,  120,  120,  120,  120,  120,  120,  120,  125,  125,  125,  125,  120,  115,  110,  110,  110,  110 },
{ 108,  108,  108,  108,  115,  115,  115,  115,  115,  115,  115,  115,  115,  110,  110,  110,  110,  110 },
{ 100,  100,  100,  104,  104,  104,  105,  107,  107,  107,  107,  107,  107,  107,  107,  107,  107,  107 },
{ 90, 90, 90, 86, 88, 88, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94  },
{ 85, 85, 85, 79, 80, 75, 83, 86, 86, 86, 86, 86, 86, 86, 86, 86, 86, 86  },
{ 75, 75, 75, 73, 72, 73, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75  },
{ 68, 68, 68, 68, 65, 65, 70, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75  },
{ 64, 64, 64, 63, 63, 63, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64  },
{ 46, 46, 46, 46, 46, 51, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60  }

};

int f1Rpm[18] = { 0, 560, 1120, 1680, 2240, 2800, 3360, 3920, 4480, 5040, 5600, 6160, 6720, 7280, 9500, 9600, 9700, 9800 };
int f1Map[18] = { 14, 28, 42, 56, 70, 84, 98, 112, 126, 140, 154, 168, 182, 196, 240, 250, 260, 270 };
```

Slika 47. Matrica volumetrijske efikasnosti



Slika 48. 3D prikaz matrice volumetrijske efikasnosti

Retci u tablici predstavljaju tlakove, a stupci brzine vrtnje. Tlakovi i brzine vrtnje za pojedini redak i stupac podesivi su i pohranjeni u jednodimenzijskim nizovima *f1Rpm*[18] i *f1Map*[18], a vrijednosti između određuju se linearnom interpolacijom. Način određivanja pozicije u tablici prema trenutnoj brzini vrtnje i tlaku prikazan je na slici 49. Prva *for* petlja redom uspoređuje trenutni tlak u usisnom kolektoru s članovima niza f1Map. Kada naiđe na član čija je vrijednost veća od trenutnog tlaka(redak 12) uzima da se tlak nalazi između trenutnog i prethodnog člana. Prema udaljenosti od pojedinog člana izračunava se pozicija člana Y koja se kasnije koristi kod interpoliranja volumetrijske efikasnosti. Isti postupak vrijedi i za brzinu vrtnje gdje se dobiva pozicija X.

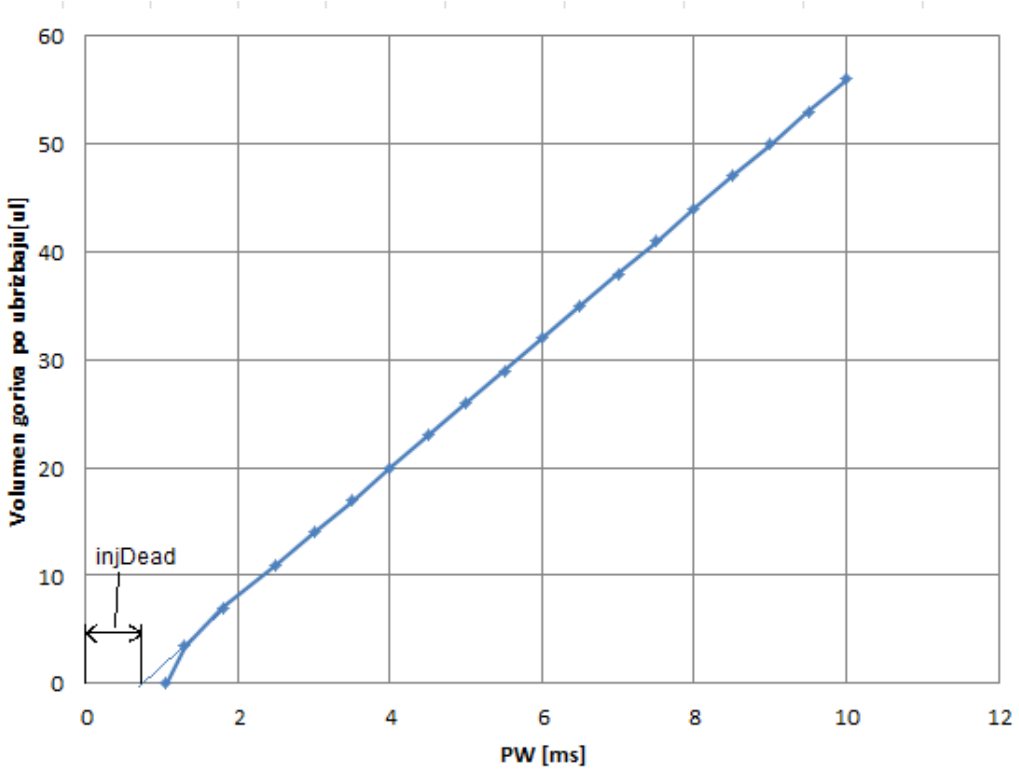
```
8  if ( fuelMap==1 ){   ////////////////////////////Fuel map 1
9
10 for ( i=0 ; i<18 ; i++ ){
11     float y1,y2;
12       if ( kPa<f1Map[i] ){
13         if ( kPa<f1Map[0] ){ Y=0; }
14         else{
15           y1=kPa-f1Map[i-1];
16           y2=f1Map[i]-f1Map[i-1];
17           Y=y1/y2+i-1;
18           }
19         break;
20       }
21   }
22   if ( kPa>=f1Map[17] ){ Y=16.99; }
23
24 for ( i=0 ; i<18 ; i++ ){
25    float x1,x2;
26     if ( RPM<f1Rpm[i] ){
27       if ( RPM<f1Rpm[0] ){ X=0; }
28       else{
29         x1=RPM-f1Rpm[i-1];
30         x2=f1Rpm[i]-f1Rpm[i-1];
31         X=x1/x2+i-1;
32         }
33       break;
34     }
35   }
36   if ( RPM>=f1Rpm[17] ){ X=16.99; }
37
38 }
```

Slika 49. Određivanje volumetrijske efikasnosti prema trenutnoj radnoj točki

Nakon toga se određuju članovi u tablici volumetrijske efikasnosti između kojih se izvodi linearna interpolacija. Ako je npr. X=2.4 i Y=2.2 interpolacija se radi između članova *f1*[2][2], *f1*[3][2], *f1*[2][3] i *f1*[3][3]. Prvo se interpoliraju članovi u istim retcima, a zatim se

interpolirane vrijednosti međusobno interpoliraju čime se dobiva konačna volumetrijska efikasnost za trenutnu radnu točku.

Na izračunatu duljinu ubrizgavanja dodaje se mrtvo vrijeme brizgaljke(*injDead*) koje predstavlja dio vremena od zadane duljine impulsa za koje brizgaljka ne ubrizgava gorivo. Time se postiže da je zadano vrijeme ubrizgavanja jednako stvarnom vremenu ubrizgavanja.



Slika 50. Karakteristika brizgaljke

Graf na slici 50. prikazuje ovisnost količine ubrizganog goriva o duljini otvorenosti brizgaljke. Ubrizgavanjem goriva u menzuru određen broj puta(npr. 1000) i zatim dijeljenjem volumena goriva s brojem ubrizgavanja dobije se volumen goriva po ubrizgaju. Ponavljanjem mjerenja za nekoliko različitih duljina ubrizgavanja dobiva se graf kao na slici 50. Karakteristika brizgaljke ovisi o naponu akumulatora pa je mjerenje potrebno ponoviti za nekoliko različitih napona. Mjesto gdje bi linearni dio grafa sjekao x os je mrtvo vrijeme brizgaljke, a dobivena vremena unose se u *look up* tablicu prema kojoj se linearnom interpolacijom izračunava mrtvo vrijeme za trenutni napon.

Duljina ubrizgavanja dobivena jednadžbom (32) vrijedi za stacionarne uvjete(zaklopka miruje) i normalnu radnu temperaturu motora. U fazi hladnog starta, zagrijavanja ili nestacionarnih uvjeta rada, osnovnoj duljini otvorenosti brizgaljke pridodaju se korekcijski

faktori koji osiguravaju održavanje željenog faktora pretička zraka. Kod *speed density* metode temperatura zraka također je kritičan faktor jer promjena temperature utječe na promjenu gustoće zraka, a time i na masu zraka koja ulazi u cilindru. Korekcijski faktori koji se dodaju na osnovnu duljinu impulsa su:

- korekcija hladnog starta(*crankEnrich)*
- korekcija nakon starta(*afterStartEnrich*)
- korekcija kod zagrijavanja(*warmUpEnrich*)
- korekcija zbog promjene temperature zraka(*IATcorr*)
- korekcija uslijed tranzijentnih pojava(*accEnrich*)

Svi korekcijski faktori množe se s osnovnom duljinom impulsa bez mrtvog vremena brizgaljke, a konačni izraz glasi:

$$PW = reqPW \cdot \frac{VE}{100} \cdot \frac{MAP}{101} \cdot tempEnrich + accEnrich + injDead \quad \textbf{(33)}$$

accEnrich je dodatak kod tranzijentnih pojava, a tempEnrich umnožak korekcija na koje utječe temperatura:

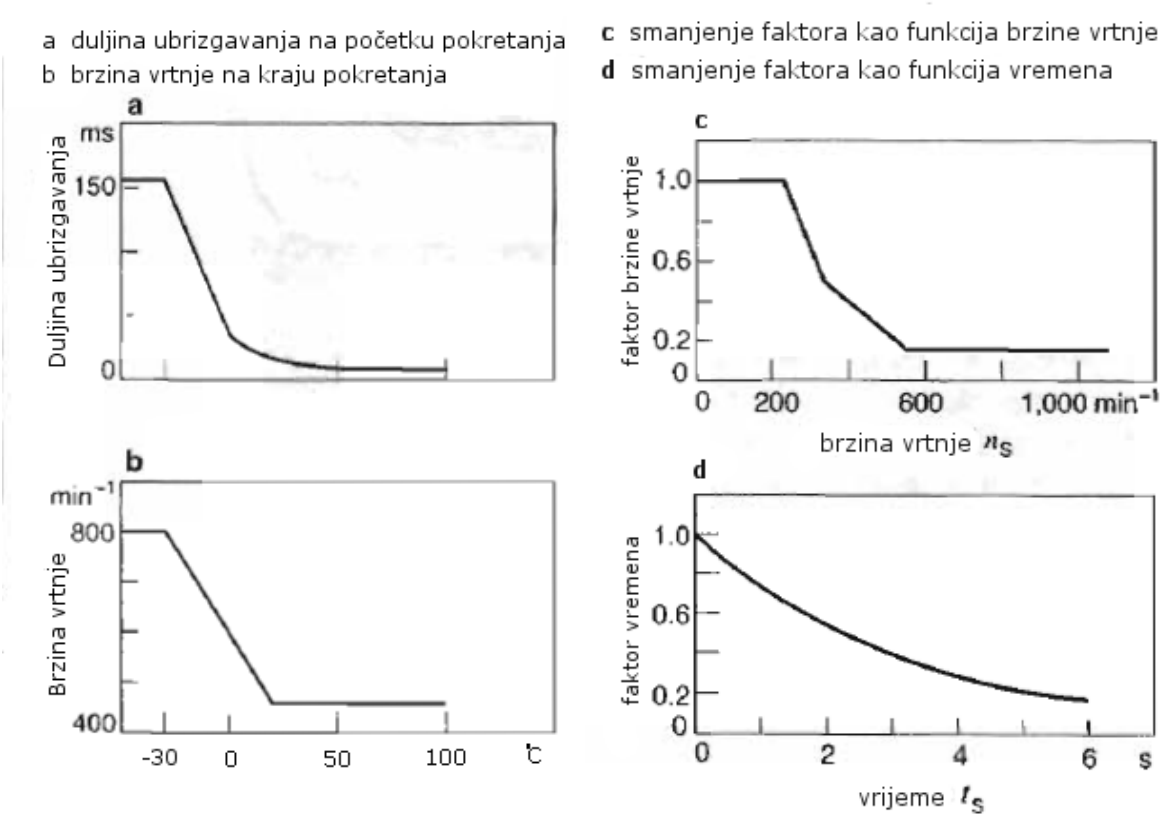
$$tempEnrich = crankEnrich \cdot afterStartEnrich \cdot warmUpEnrich \cdot IATcorr \quad \textbf{(34)}$$

Korekcija hladnog starta(*cranking enrichment*)

Kod pokretanja hladnog motora isparavanje goriva je ograničeno zbog:

- hladnog zraka u usisu
- hladnih stijenki usisnih kanala
- hladnih stijenki cilindara i komore izgaranja
- visokog tlaka i male brzine strujanja u usisnoj cijevi(vrijedi i kod toplog pokretanja)

Navedeni faktori utječu na povećanu kondenzaciju dijela ubrizganog goriva na stjenke usisnih kanala i ventila što dovodi do smanjenja količine goriva koja ulazi u cilindar. Ubrizgavanjem viška goriva u ovoj fazi osigurava se da kondenzacija završi što prije i da dovoljna količina goriva stigne u cilindar. Kondenzacija najviše ovisi o temperaturi stijenki usisnih kanala i ventila pa je sukladno tome korekcija ubrizgavanja zadana u odnosu na temperaturu motora tj. rashladne tekućine(Slika 51., graf a). Osim temperature, utjecaj na kondenzaciju ima i brzina strujanja zraka u usisnom kanalu. Pri većim brzinama vrtnje strujanje zraka je brže, a kondenziranje goriva slabije pa se faktor obogaćivanja smjese smanjuje(graf c). Ova faza prestaje kada brzina vrtnje naraste iznad zadane vrijednosti ovisne o temperaturi(graf b). Iznad te brzine vrtnje smatra se da je motor pokrenut i korekcija kod starta se prekida. U dijagramima su prikazane samo okvirne vrijednosti korekcijskih faktora i nisu iste za svaki motor.

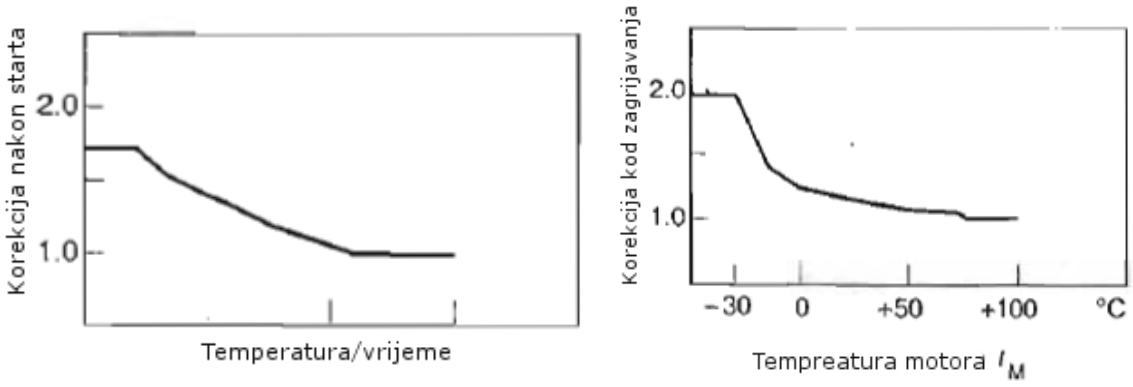


Slika 51. Korekcijski faktori kod hladnog starta[1]

Korekcija nakon starta i tijekom zagrijavanja

Odmah nakon pokretanja, kada brzina vrtnje naraste iznad zadane, duljina ubrizgavanja počinje se određivati prema glavnoj mapi(VE). Obogaćivanje smjese još je uvijek potrebno zbog hladnih stijenki cilindra i usisnih kanala. Prvi nekoliko sekundi nakon starta korekcijski

faktor uvećava se u odnosu na onaj korišten kod zagrijavanja motora. Povećanje korekcijskog faktora ovisi o temperaturi rashladne tekućine i smanjuje se s vremenom(slika 52, lijevo) ili brojem ciklusa nakon starta. Ovisnost korekcije o broju ciklusa ima više smisla jer se time uzima u obzir da se motor prije zagrije radeći 5 sekundi pri 2000min^{-1} nego pri 1000min^{-1} s obzirom da napravi dvostruko veći broj ciklusa. Od tada počinje vrijediti samo faktor zagrijavanja(slika 52, desno) također ovisan o temperaturi koji se smanjuje sve do postizanja radne temperature.



Slika 52. Korekcijski faktori nakon starta i tijekom zagrijavanja[1]

Korekcija zbog promjene temperature zraka

Ovim faktorom kompenzira se promjena gustoće zraka zbog promjene temperature. Razlog zašto se mjerena temperatura ne unosi u jednadžbu idealnog plina i izravno korigira masu zraka u cilindru je što potrebna korekcija ovisi o položaju senzora temperature i ne mora nužno pratiti jednadžbu. Korekcija se očitava iz *lookup* tablice u ovisnosti o temperaturi zraka i moguće ju je podesiti prema potrebi.

Korekcijski faktori pohranjeni su u jednodimenzijskim nizovima s 8 elemenata. Svaki element predstavlja jednu temperaturu ili neku drugu fizikalnu veličinu, ovisno o kojem korekcijskom faktoru se radi. Redni broj elementa povezan je s vrijednosti fizikalne veličine preko jednadžbe pravca. Princip određivanja jednadžbe isti je kao i kod senzora temperature(jednadžba 21). Svi navedeni korekcijski faktori zajedno s pripadajućim jednadžbama i okvirnim korekcijama prikazani su u tablici 5.

Tablica 6. Korekcijski faktori ubrizgavanja

Sve korekcije ovisne o temperaturi	Element niza	0	1	2	3	4	5	6	7
	Temperatura[°C]	-20	-5	10	25	40	55	70	85
	Korekcija[%]	180	160	150	130	125	115	100	100
	Jednadžba	y=0.06667x+1.333							
Korekcija za broj ciklusa nakon starta (afterStartCycles)	Element niza	0	1	2	3	4	5	6	7
	Broj ciklusa	0	15	30	45	60	75	90	105
	Korekcija[%]	180	160	150	130	125	115	100	100
	Jednadžba	y=0.06667x							
Faktor brzine vrtnje kod pokretanja (crankSpeedCorr)	Element niza	0	1	2	3	4	5	6	7
	Brzina vrtnje	0	120	240	360	480	600	720	840
	Korekcija[%]	100	100	100	80	50	20	0	0
	Jednadžba	y=0.008333							
Faktor vremena kod pokretanja (crankTimeCorr)	Element niza	0	1	2	3	4	5	6	7
	Vrijeme[s]	0	1	2	3	4	5	6	7
	Korekcija[%]	100	100	80	50	50	50	50	50
	Jednadžba	y=x							

Ukupna korekcija kod pokretanja jednaka je:

$$crankEnrich = crankTempEnrich \cdot crankSpeedCorr \cdot crankTimeCorr \quad \textbf{(35)}$$

Ukupna korekcija nakon pokretanja jednaka je :

$$afterStartEnrich = afterStartTemp \cdot afterStartCycles \quad \textbf{(36)}$$

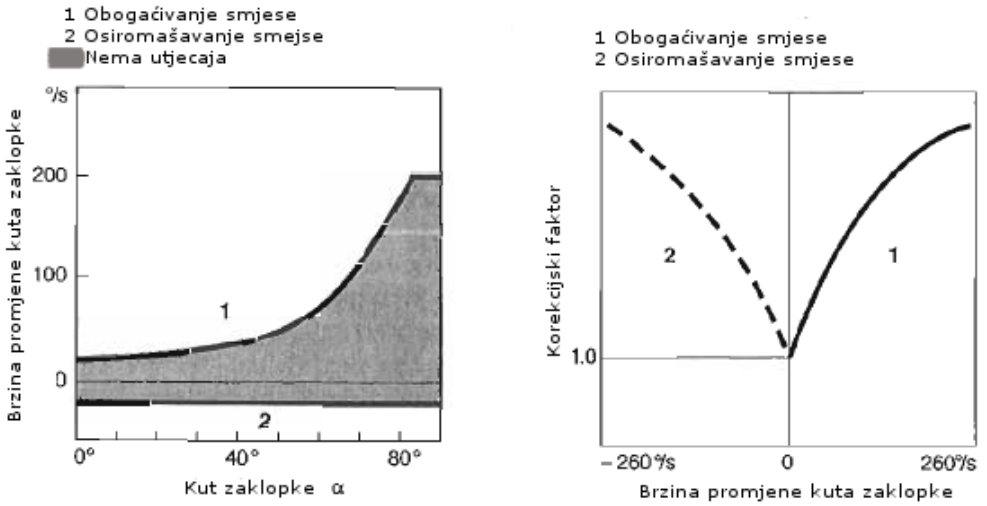
Korekcija kod tranzijentnih pojava

Pod tranzijentnim pojavama smatra se promjena opterećenja uslijed promjene kuta zaklopke koja zahtjeva korekciju u količini ubrizganog goriva s ciljem održavanja konstantnog faktora pretička zraka. Glavni razlog potrebe za korekcijom je ovisnost kondenzacije goriva na stjenke usisnih kanala o tlaku u usisnoj cijevi. Pri niskom tlaku, kada je opterećenje malo, gorivo u usisnoj cijevi je gotovo u potpunosti ispareno i samo mali dio se kondenzira na stjenke. Kada tlak naraste(slučaj otvaranja zaklopke ili pada brzine vrtnje), isparavanje goriva slabi, a sloj goriva na stjenkama se povećava. Rezultat je poremećaj ravnoteže kondenziranja i

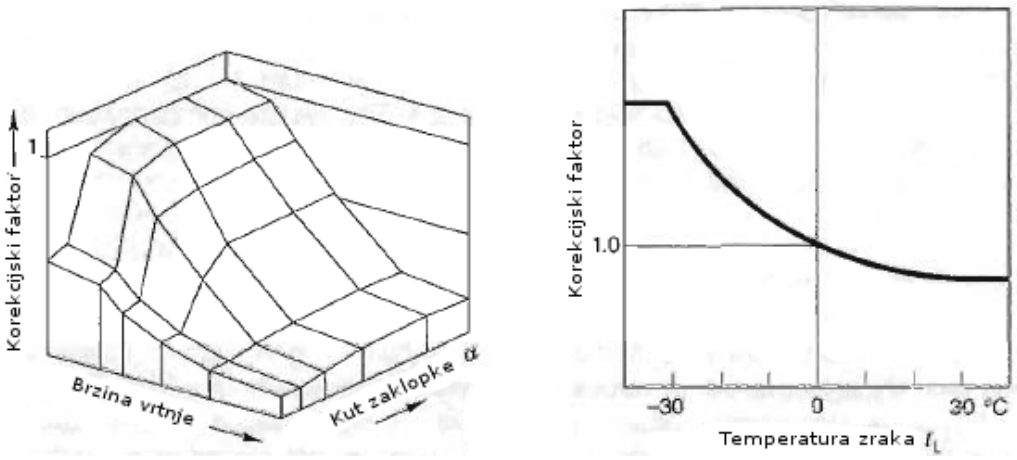
isparavanja zbog čega je kod otvaranja zaklopke potrebno ubrizgati dodatnu količinu goriva dok se ne poveća kondenzirani sloj i ponovo uspostavi ravnoteža. Suprotno se događa kod zatvaranja zaklopke, kada se mora ubrizgavati manje goriva zbog isparavanja određene količine sa stijenki usisnih kanala. Temperatura također ima veliki utjecaj na isparavanje. Kod hladnih stijenki ili hladnog zraka u usisu sloj kondenziranog goriva je veći pa i korekcija ubrizgavanja mora biti veća.

Drugi problem koji se javlja kod tranzijentih pojava je mjerenja tlaka u usisnom kolektoru. Kao što je opisano u 4.3.2., tlak u usisu se mijenja tijekom ciklusa zbog otvaranja i zatvaranja usisnih ventila. Da bi se dobio stabilan tlak za izračunavanje duljine ubrizgavanja, signal se na neki način mora filtrirati. Filtriranje je riješeno uzorkovanjem signala jednom u ciklusu tj. onoliko puta koliko ima cilindara i to svaki puta na istom kutu zakreta koljenastog vratila. Na taj način se dobiva stabilan tlak bez obzira na varijaciju tlaka tijekom ciklusa. Problem koji se javlja bilo kod niskopropusnog filtriranja ili kod uzorkovanja jednom u ciklusu je kašnjenje. Ako se tlak mjeri uvijek u istoj poziciji, u sredini takta usisa, a ubrizgavanje počinje i završava prije otvaranja usisnog ventila, što se najčešće radi zbog boljeg isparavanja goriva, znači da je zadnji uzorak tlaka uzet prije više od jednog okretaja. Pri brzini vrtnje od $2000 min^{-1}$ jedan okretaj traje 30ms što je više nego dovoljno da se tlak u usisu značajno promjeni(vozač može pritisnuti pedalu gasa od 0 do 100% za oko 80ms). Ubrizgana količina goriva će u tom slučaju odgovarati tlaku izmjerenom u prošlom ciklusu. Čak i kada bi se tlak mjerio prije početka ubrizgavanja, do kraja usisnog takta opet se može promjeniti. Zato se tlak mjeri u sredini takta usisa i uspoređuje sa prošlim izmjerenim tlakom, ako je razlika veća od neke odabrane, brizgaljka se otvara još jedan put i ubrizgava preostalu količinu goriva.

Korekcija smjese zbog kondenzacije goriva na stjenkama može se riješiti definiranjem nekoliko *lookup* tablica prema kojima se dodatna količina goriva određuje u ovisnosti o brzini promjene kuta zaklopke, temperaturi zraka i rashladne tekućine, brzini vrtnje i kutu zaklopke. Utjecaj navedenih parametara na korekciju prikazan je u sljedećim dijagramima.



Slika 53. Utjecaj brzine promjene kuta zaklopke na korekciju ubrizgavanja[1]

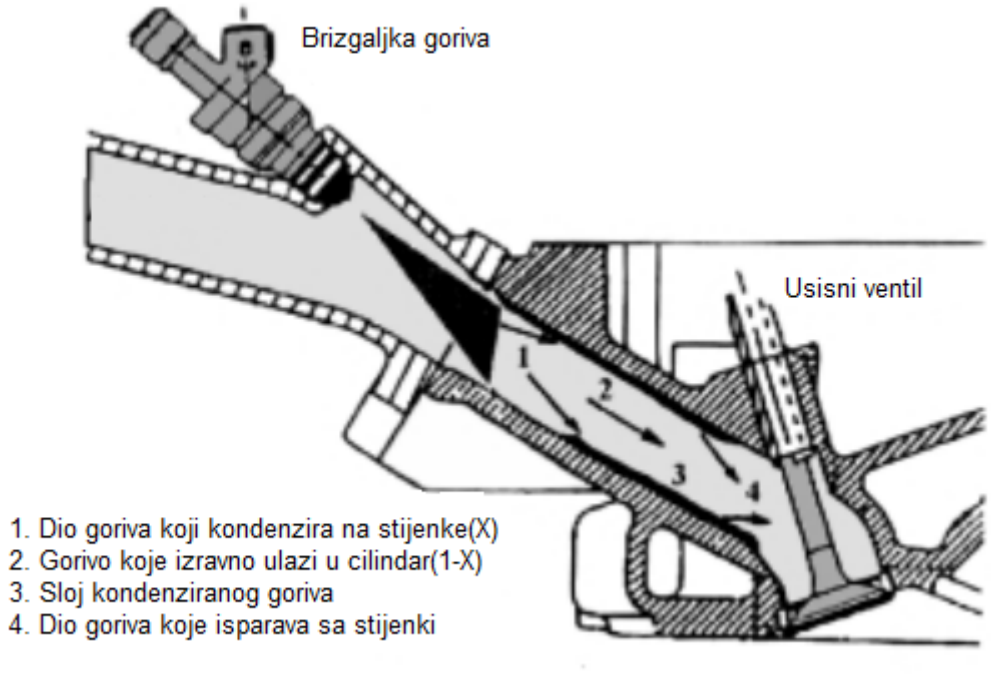


Slika 54. Utjecaj brzine vrtnje, kuta zaklopke i temperature zraka na korekciju ubrizgavanja[1]

Problem kod ovog načina korekcije je što nema direktne poveznice sa stvarnim modelom motora. Svaki parametar podešava se zasebno i potrebno je puno vremena da bi se podesili svi mogući slučajevi za različite brzine otvaranja zaklopke, temperature i brzine vrtnje. Kao što se vidi na slici 54., brzina promjene kuta zaklopke za aktiviranje korekcije ovisi o trenutnom kutu. Kod zatvorene zaklopke, ista promjena kuta dovesti će do većeg povećanja tlaka u usisu nego kod otvorene zaklopke. Zato se kod otvorene zaklopke korekcija aktivira na puno većim brzinama promjene kuta. Što je brzina otvaranja veća, potrebna je veća korekcija ubrizgavanja s time da veza nije linearna. Brzina vrtnje utječe na brzinu strujanja zraka u usisnoj cijevi zbog čega se mijenja količina goriva koja kondenzira na stjenkama. Porastom brzine vrtnje brzine strujanja su veće, kondenziranje slabije i potrebna korekcija manja. Utjecaj temperature

na korekciju je logičan, porastom temperature zraka i stijenki slabi kondenzacija goriva, a time i korekcijski faktor.

Druga metoda kompenzacije implementirana u programu temelji se na relativno jednostavnom x-Tau[10] modelu koji opisuje stvarne pojave dinamike goriva u usisu. Kada se gorivo ubrizga u usisnu cijev, jedan dio miješa se sa zrakom i ulazi u cilindar, a ostatak kondenzira na stijenke usisnih kanala. U isto vrijeme dio goriva sa stijenki isparava i pridodaje se gorivu koje ulazi u cilindar. Isparavanje ovisi o trenutnoj masi kondenziranog goriva, tlaku u usisnoj cijevi i temperaturi zraka i stijenki. Što se više goriva ubrizgava, više ga kondenzira na stjenke. Zato se kod nagle promjene opterećenja dio goriva troši na povećanje kondenziranog sloja te dolazi do narušavanja ravnoteže isparavanja i kondenziranja. U kratkom periodu nakon promjene opterećenja potrebno je ubrizgavati dodatnu količinu goriva koja će nadomjestiti dio kondenziran na stjenke.



Slika 55. Prikaz dinamike ubrizgavanja goriva u usisnoj cijevi[8]

Fizikalne veličine koje će se koristiti u sljedećim jednadžbama su:

- m_u – ubrizgana masa goriva
- $m_{c1,2}$ – masa goriva koja ulazi u cilindar
- m_k – masa goriva koje se kondenzira na stjenke

- $m_{k,u}$ – ukupna masa goriva na stjenkama
- X – postotak goriva koje kondenzira
- *Tau* – vremenska konstanta isparavanja, vrijeme potrebno da ukupna masa kondenziranog goriva ispari
- *Dt* – vrijeme između dva ubrizgavanja

Kod svakog ubrizgavanja dio goriva ulazi direktno u cilindar:

$$m_{c1} = m_u \cdot (1 - X) \tag{37}$$

Dio goriva se kondenzira i pridonosi povećanju mase sloja na stjenkama:

$$m_k = m_u \cdot X \tag{38}$$

Gorivo koje je isparilo od prošlog ubrizgavanja ulazi u cilindar zajedno sa m_{c1}:

$$m_{c2} = \frac{m_{k,u}}{\frac{Tau}{dt}} \tag{39}$$

Iz navedenih jednadžbi slijedi ukupna masa goriva koja ulazi u cilindar:

$$m_{c1,2} = m_u \cdot (1 - X) + \frac{m_{k,u}}{\frac{Tau}{dt}} \tag{40}$$

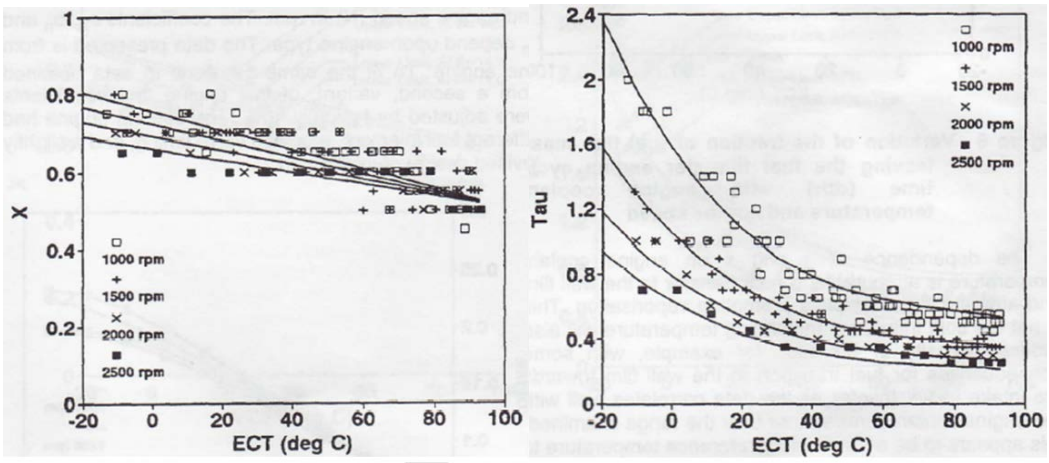
Trenutna masa kondenziranog goriva jednaka je razlici novo kondenziranog i isparenog goriva:

$$m_{k,u} = m_u \cdot X - \frac{m_{k,u}}{\frac{Tau}{dt}} \tag{41}$$

Korisno je iz jednadžbe (41) izraziti masu ubrizganog goriva jer o njoj ovisi duljina otvorenosti brizgaljke:

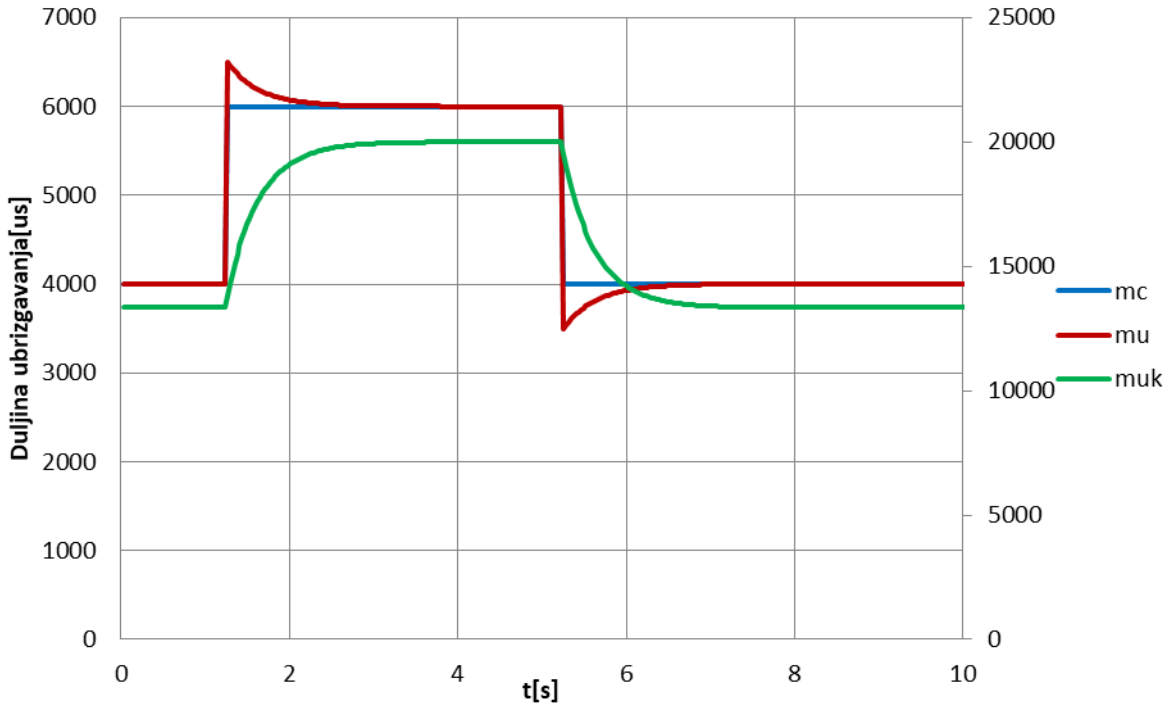
$$m_u = \frac{m_{c1,2} - \frac{m_{k,u}}{\frac{Tau}{dt}}}{(1-X)} \tag{42}$$

$m_{c1,2}$ je količina goriva koja mora ući u cilindar, a dobiva se iz jednadžbe idealnog plina i volumetrijske efikasnosti kako je prikazano na početku poglavlja. U stacionarnim uvjetima m_u je jednak $m_{c1,2}$. Parametri modela X i Tau nisu konstantni već se mijenjaju s uvjetima rada, a najveći utjecaj na parametre ima temperatura stijenki tj. rashladne tekućine i brzina vrtnje. Dijagrami prikazani na slici 56. mogu se koristiti kao smjernica kod podešavanja parametara.



Slika 56. Utjecaj temperature rashladne tekućine na faktore X i Tau[11]

Masa ubrizganog goriva proporcionalna je s duljinom ubrizgavanja u linearnom području brizgaljke pa ih je moguće međusobno zamijeniti u jednadžbi (42). Dijagram na slici 57. prikazuje utjecaj skokovite promjene zadane duljine ubrizgavanja $m_{c(}$zbog nagle promjene opterećenja), na potrebnu duljinu ubrizgavanja m_u da bi zadana količina goriva dospjela u cilindar. Zelena linija predstavlja masu goriva na stjenkama, ali prikazana preko duljine ubrizgavanja. Vidljivo je da se kod nagle promjene zadane duljine ubrizgavanja potrebna duljina kratkotrajno povećava sve dok se ne izgradi odgovarajući sloj goriva na stjenkama i ponovo postigne ravnoteža.



Slika 57. Odaziv X-Tau modela

Komunikacija s integriranim krugom MC33810

Princip rada integriranog kruga MC33810 objašnjen je u prethodnom poglavlju, a sada će se pokazati njegovo podešavanje preko SPI komunikacije. Naredbe su definirane u grupama poruka duljine 16 bitova, gdje prva četiri bita predstavljaju adresu tj. postavku koja se želi podesiti, a idućih 12 bitova naredbe za tu postavku. Sve adrese i naredbe nalaze se u *datasheetu* proizvođača, a pokazat će se samo ona koja se trenutno koristi. Princip podešavanja svih ostalih naredbi je identičan.

Uređaj je potrebno podesiti kod svakog pokretanja jer se gašenjem postavke vraćaju na tvorničke. Jedina postavka koja ne odgovora je *Igintion* mod rada koji treba promijeniti u GPGD mod. To se izvodi slanjem naredbe prikazane u tablici 7 preko SPI-a:

Tablica 7. Naredba za promjenu moda rada

Naredba	**Adresa**				**Naredba**											
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Odabir moda rada	0	0	0	1	1	1	1	1	0	0	x	x	0	0	0	0
					IGN/GPGD mod(0/1)				V10				PWM ON/OFF(1/0)			

Bitovi 15-12 određuju postavku koja se želi podesiti, što je u ovom slučaju odabir moda rada. Sljedeća četiri bita definiraju sam mod rada, a svaki bit odnosi se na jedan od četiri izlaza. Podešavanjem ovih bitova na 1 odabire se GPGD, a na 0 IGN mod rada. Ostali bitovi ostavljaju se na defaultnim vrijednostima jer se te funkcije ne koriste.

Druga korištena naredba je očitavanje grešaka kratkog spoja i otvorenog kruga kod brizgaljki. Slanjem naredbe za čitanje registara dobiva se odgovor u obliku poruke duljine 16 bitova gdje svaki bit predstavlja jednu grešku. Ako je bit 1 znači da ima greške.

Tablica 8. Naredba za očitavanje greške

Naredba	Adresa				Naredba											
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Čitanje registara	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0
									Adresa greške							

Tablica 8. prikazuje poruku za čitanje greške. Bitovi 7-4 određuju koja greška se želi očitati, a adresa 0001 prikazana u tablici označava čitanje greške za izlaze OUT0 i OUT1. Odgovor na poruku je sljedeći:

Tablica 9. Odgovor na naredbu čitanja greške

Naredba	Adresa				Naredba											
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT1 OUT0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
greška									greška OUT1				greška OUT0			

Gdje bitovi 7-4 označavaju grešku za OUT1, a bitovi 3-0 za OUT0. Bitovi 7 i 3 javljaju grešku pregrijavanja *drivera*, bitovi 6 i 2 kratki spoj, bitovi 5 i 1 otvoreni krug ako je brizgaljka zatvorena i bitovi 4 i 0 otvoreni krug ako je zadano da brizgaljka bude otvorena. Očitane greške moguće je poslati na računalo i tako olakšati korisniku otkrivanje kvarova bez potrebe za ručnim ispitivanjem.

4.3.7. Paljenje smjese

Glavni parametar kod paljenja smjese je trenutak preskakanja iskre koji se definira kao udaljenost klipa od GMT u stupnjevima zakreta koljenastog vratila. Pozitivnim kutom označava se paljenje prije GMT i najčešće se naziva kut pretpaljenja. Kut pretpaljenja ima veliki utjecaj na porast tlaka u cilindru, potrošnju goriva, snagu, zagrijavanje, emisije ispušnih plinova i pojavu detonantnog izgaranja. Optimalan kut ovisi o trenutnoj radnoj točki, a

najveći utjecaj imaju faktor pretička zraka, opterećenje, brzina vrtnje, temperatura rashladne tekućine i temperatura zraka. Glavna *lookup* tablica prema kojoj se određuje kut pretpaljenja je dimenzija 18x18 u kojoj stupci predstavljaju brzine vrtnje, a retci tlakove u usisnom kolektoru. Očitavanje vrijednosti iz tablice temelji se na istom principu kao i kod tablice volumetrijske efikasnosti. Uz glavnu tablicu postoje još dvije dimenzija 1x8 u koje se unosi korekcija kuta pretpaljenja kao funkcija temperature zraka i rashladne tekućine te jedna tablica za određivanje duljine punjenja bobine u ovisnosti o naponu.

Točan trenutak paljenja određuje se na sljedeći način. Najprije se očitava kut paljenja za trenutnu radnu točku, dodaju se korekcije ako ih ima i računa se vrijeme potrebno da se koljenasto vratilo okrene za očitani kut pri trenutnoj brzini vrtnje.

$$T_{fi} = \frac{(fi + fiCorr)}{toothGap} \cdot RPMsplit \tag{43}$$

- *RPMsplit* – vremenska udaljenost dva zuba
- *toothGap* – kutna udaljenost dva zuba
- *fi* – kut pretpaljenja
- *fiCorr* – korekcija kuta ako se GMT ne poklapa točno sa zubom

Zatim se određuje prvi zub prije početka punjenja i prvi zub prije paljenja. Na ovom zubu će se kasnije podesiti brojač koji će za izračunato vrijeme upaliti ili ugasiti bobinu. Prvo se izračunava udaljenost tog zuba od GMT, zaokruženo na veću vrijednost, a zatim njegov redni broj.

$$chargeNZ = \frac{fi + fiCorr}{toothGap} + \frac{dwell}{RPMsplit} + 2 \tag{44}$$

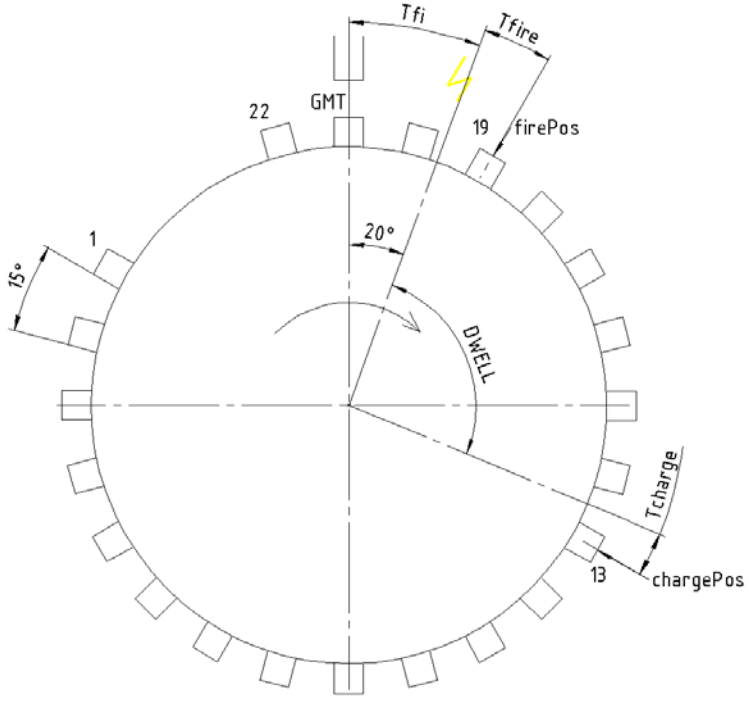
$$chargePos = GMT1 - chargeNZ \tag{45}$$

$$fireNZ = \frac{fi + fiCorr}{toothGap} + 1 \tag{46}$$

$$firePos = GMT1 - fireNZ \tag{47}$$

- chargeNZ - broj zuba prije GMT na kojem se podešava brojač za punjenje bobine, 2 se dodaje zato što su sve varijable tipa *integer* pa se dijeljenjem zaokružuju na manji broj
- chargePos – zub na kojem se podešava brojač za punjenje bobine, ako je rezultat negativan broj na njega se dodaje ukupan broj zuba na enkoderu
- GMT1 – zub koji se nalazi ispred senzora u GMT
- fireNZ – broj zuba prije GMT na kojem se podešava brojač za gašenje bobine
- firePos – zub na kojem se podešava brojač za gašenje bobine

Na slici 58. pokazan je primjer za 24-2 enkoder koji se koristi na motoru Strixa. U primjeru je zadan kut pretpaljenja 20° i duljina punjenja 3ms pri 4800min^{-1}(razmak između 2 zuba 520us).



Slika 58. Primjer enkodera s prikazanim početkom punjenja bobine i preskakanja iskre

Iz prethodnih jednadžbi slijedi:

$$T_{fi} = \frac{(20+0)}{15} \cdot 520 = 693us \qquad \textbf{(48)}$$

$$chargeNZ = \frac{20+0}{15} + \frac{3000}{520} + 2 = 8 \quad \textbf{(49)}$$

$$chargePos = 21 - 8 = 13 \quad \textbf{(50)}$$

$$fireNZ = \frac{20+0}{15} + 1 = 2 \quad \textbf{(51)}$$

$$firePos = 21 - 2 = 19 \quad \textbf{(52)}$$

Sada kada su poznati zubi na kojima se zadaje brojač, treba odrediti vrijeme od zuba do početka punjenja ili pražnjenja. Točna pozicija KV poznata je samo kada zub prolazi pored senzora tj. svakih 15°, sve pozicije između određuju se vremenskom udaljenosti na temelju zadnje poznate brzine vrtnje i kutne udaljenosti. Vremenska udaljenost zuba *firePos* od pozicije trenutka paljenja je:

$$Tfire = RPMsplit \cdot fireNZ - T_{fi} = 520 \cdot 2 - 693 = 347 \quad \textbf{(53)}$$

Vremenska udaljenost zuba *chargePos* od pozicije početka punjenja:

$$Tcharge = RPMsplit \cdot chargeNZ - T_{fi} - dwell = 520 \cdot 8 - 693 - 3000 = 467 \quad \textbf{(54)}$$

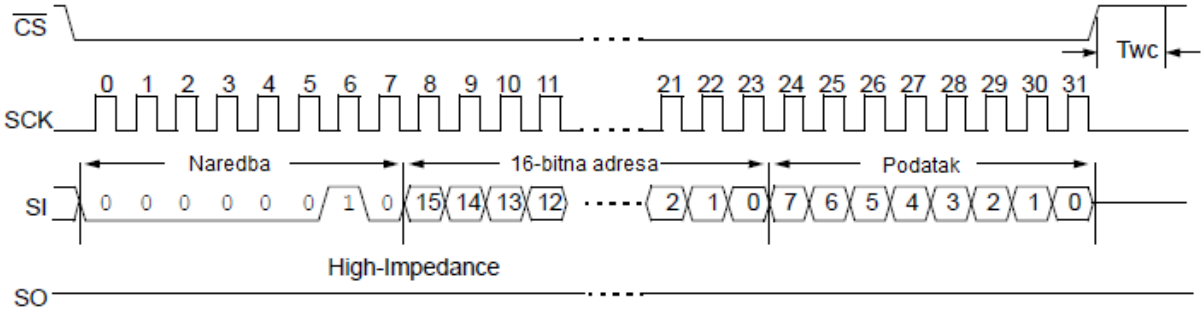
Kada se u prekidnoj funkciji detektira zub *chargePos* ili *firePos*, podešava se brojač koji će generirati prekid za vrijeme *Tcharge* odnosno *Tfire*.

4.3.8. Komunikacija s EEPROM memorijom

EEPROM 25LC640 koristi SPI komunikacijski protokol za čitanje i pisanje podataka. EEPROM može pohraniti 8192 bajta, a svaki bajt ima vlastitu adresu u memoriji. U jedan bajt može se pohraniti vrijednost od 0 do 255, a sve veće vrijednosti moraju se rastaviti na više bajtova.

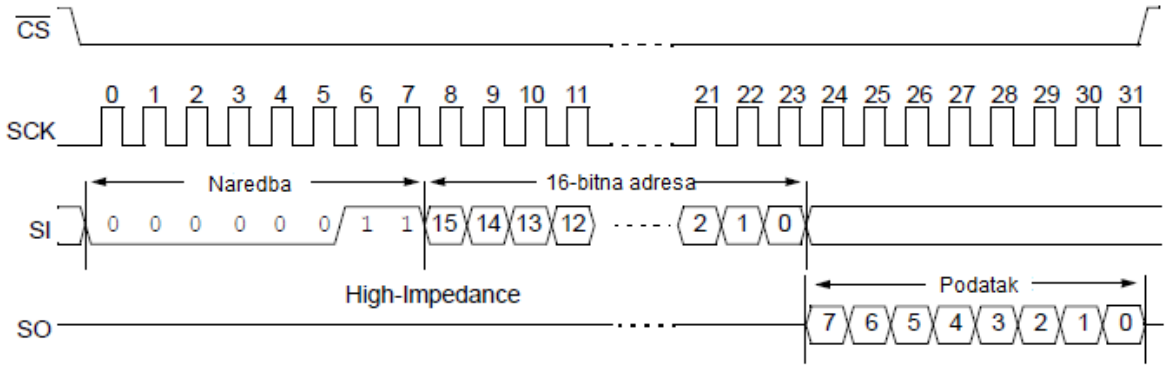
Zapisivanje u EEPROM započinje postavljanjem CS linije na nisku razinu. Nakon toga šalje se WREN(*write enable*) naredba kojom se dozvoljava pisanje u EEPROM. Sve naredbe

opisane su u uputstvima proizvođača, a naredba za WREN je bajt *0000 0110*. Kada je bajt poslan, CS linija ponovo se diže na visoku razinu i pisanje se omogućuje. Zatim se CS ponovo spušta i šalje se 8-bitna naredba za pisanje(*0000 0010*) iza čega slijedi 16-bitna adresa bajta koji se želi zapisati(Slika 58). Prva tri najznačajnija bita se ne koriste jer EEPROM sadrži samo 2^{13} adresa(8192). Nakon adrese šalje se bajt koji se želi pohraniti, a moguće je poslati do 32 bajta za redom bez prekida. Kada su željeni bajtovi poslani, CS linija diže se na visoku razinu i kreće zapisivanje poslanih bajtova u memoriju. Zapisivanje može trajati do 5ms unutar kojih se ne smiju slati novi podatci.



Slika 59. Izgled poruke za pisanje podataka u EEPROM

Čitanje bajtova obavlja se na sličan način kao i pisanje. Najprije se šalje naredba čitanja *0000 0011*, zatim adresa bajta i na kraju prazan bajt(vrijednost nije bitna) na koji EEPROM odgovara slanjem vrijednosti na toj adresi.



Slika 60. Izgled poruke za čitanje podataka iz EEPROM-a

U EEPROM-u se pohranjuju vrijednosti svih *lookup* tablica i postavki koje korisnik može mijenjati i spremati, a čitanje se obavlja pri svakom pokretanju ECU-a.

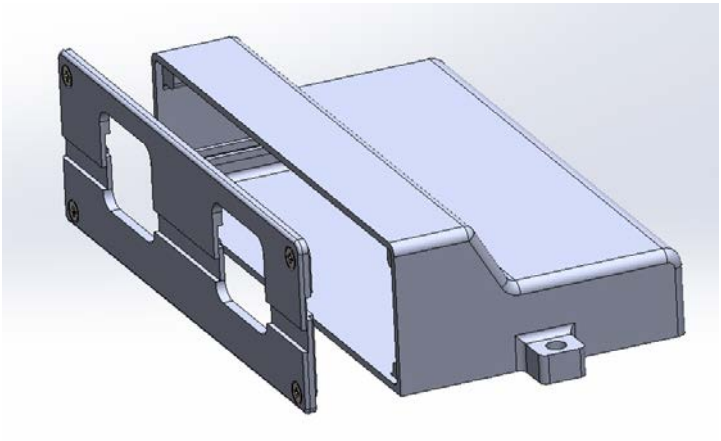
5. Kućište i konektori

Glavni zadatak kućišta je zaštita tiskane pločice od mehaničkih oštećenja i vlage uz što manju masu i dovoljnu toplinsku vodljivost zbog hlađenja komponenti. Kako se proizvodi samo jedan primjerak kućišta, ali i zbog kompleksnosti samog oblika, 3D printanje SLS postupkom odabrano je kao najbolja opcija proizvodnje. Poliamid PA12 najčešće je korišten materijal kod SLS postupka, a odlikuju ga dobra mehanička svojstva i otpornost na kemikalije. SLS postupak, zbog vrlo dobre veze između printanih slojeva, osigurava izotropna svojstva materijala i omogućuje printanje složenih i visećih oblika bez potpore.

Konektori moraju pouzdano povezivati tiskanu pločicu s okolinom. Za tu svrhu koriste se dva Tyco Amp Superseal 1.0 konektora koji osiguravaju vodonepropustan i robustan spoj pri radnoj temperaturi od -40°C do 125°C . Svaki konektor ima 34 kontakta uz najveća dozvoljenu struju 15A po kontaktu. Da bi se spriječilo krivo spajanje, konektori imaju različite položaje zareza, što onemogućuje zamjenu tj. ukapčanje lijevog konektora u desni i obratno.

Slika 61. Konektor za tiskanu pločicu(lijevo) i konektor za kabel(desno) – Tyco Amp Superseal 1.0

Slika 62. prikazuje izvedbu kućišta. Prednja ploča napravljena je iz dva dijela zbog mogućnosti montaže konektora. Ploče se na spoju preklapaju u duljini od 15mm čime se osigurava bolje brtvljenje, a dodirne plohe dodatno se premazuju silikonskim brtvilom prije sklapanja. Četiri vijka pričvršćuju ploču za kućište, a između se nalazi gumena brtva koja sprječava prodor vode.



Slika 62. Kućište

5.1. Provjera hlađenja

Sve komponentne proizvode određenu količinu topline koju treba odvesti iz kućišta. Kućište je izrađeno od polimernog materijala i nema hladnjaka koji izravno povezuje kućište i tiskanu pločicu pa se sva toplina odvodi slobodnom konvekcijom i zračenjem na zrak u kućištu te preko stijenki na okolinu. Prije izračunavanja okvirne temperature pločice potrebno je odrediti grijanje komponenti. U obzir će se uzeti samo komponente koje proizvode značajnu toplinu, a to su: regulator napona, *driver* brizgaljki MC33810, dva dvokanalna N-MOSFET-a i *driver* releja ULN2803. Proizvedena toplina kod MOSFET- računa se kao umnožak kvadrata struje i otpora R_{ds} koji daje proizvođač. Toplina koju proizvodi MC33810 za slučaj da su četiri brizgaljke konstantno otvorene:

$$\Phi_{inj} = 4 \cdot I^2 \cdot Rds = 4 \cdot 1^2 \cdot 0.2 = 0.8\text{W} \quad \textbf{(55)}$$

- I=1A – struja brizgaljke
- Rds=0.2Ω – otpor odvod-izvor MOSFET-a[15]

Toplinski tok koji proizvode dva dvokanalna MOSFET-a(ukupno četiri MOSFET-a):

$$\Phi_{gp} = 4 \cdot I^2 \cdot Rds = 4 \cdot 2^2 \cdot 0.02 = 0.32\text{W} \quad \textbf{(56)}$$

- I=2A – struja elektromagnetskih ventila
- Rds=0.02Ω – otpor odvod-izvor MOSFET-a[16]

ULN2803 je bipolarni tranzistorski niz kod kojih je zadan pad napona za određenu struju pa se proizvedeni toplinski tok računa kao umnožak struje i pada napona:

$$\Phi_{rl} = 8 \cdot V_{CE} \cdot I = 8 \cdot 1 \cdot 0.1 = 0.8\text{W} \quad \textbf{(57)}$$

- V_{CE}=1V – pad napona između kolektora i emitera za struju 100mA
- I=0.1A – tipična struja zavojnice releja upotrebljavanih u automobilima

Toplinski tok prekidačkih napajanja može se izračunati kao umnožak izlazne struje i napona pomnožen s iskoristivosti:

$$\Phi_{ps} = V_{OUT} \cdot I_{OUT} \cdot \left(\frac{1}{\eta} - 1 \right) = 5 \cdot 0.3 \cdot \left(\frac{1}{0.8} - 1 \right) = 0.375\text{W} \quad \textbf{(58)}$$

- V_{OUT}=5V – izlazni napon
- I_{OUT}=0.3A – izlazna struja(prikazana u tablici 3)

Prema tome ukupni toplinski tok koji treba odvesti je:

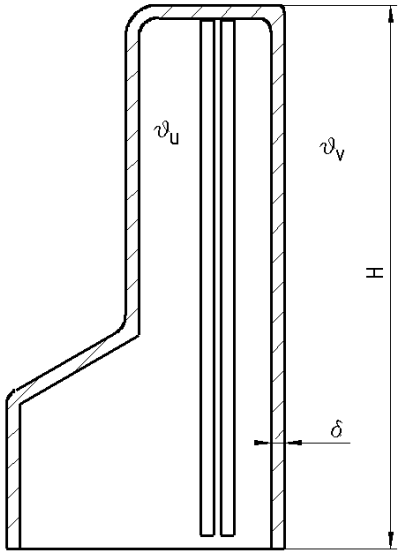
$$\Phi_{uk} = \Phi_{inj} + \Phi_{gp} + \Phi_{rl} + \Phi_{ps} = 0.8 + 0.32 + 0.8 + 0.375 = 2.3\text{W} \quad \textbf{(59)}$$

Vrijednost će se zaokružiti na 2.5W s obzirom da nisu uračunate sve komponente. Izračunati toplinski tok mora prijeći s tiskane pločice na zrak u kućištu, zatim na stjenke kućišta i na kraju u okolinu. Temperatura u kućištu u stacionarnom stanju, kada se postigne ravnoteža, izračunati će se prema izrazu za toplinski tok kroz ravnu ploču zbog pretpostavke da je kućište napravljeno od nekoliko ravnih ploča. Prijenos topline zračenjem će se zanemariti pa će dobiveni rezultat biti više na strani sigurnosti:

$$\Phi = kA(\vartheta_{\infty u} - \vartheta_{\infty v}) \quad \textbf{(60)}$$

- Φ – toplinski tok
- k – koeficijent prijelaza topline
- θ – temperatura vanjskog i unutarnjeg zraka

$$\vartheta_{\infty u} = \frac{\Phi}{kA} + \vartheta_{\infty v} \quad \textbf{(61)}$$



Slika 63. Poprečni presjek kućišta

Koeficijent prolaza topline uzima u obzir otpore zbog konvekcije na obje strane ploče i otpor provođenja kroz ploču:

$$k = \frac{1}{\frac{1}{\alpha_u} + \frac{\delta}{\lambda} + \frac{1}{\alpha_v}} \tag{62}$$

- α – koeficijent prijelaza topline
- δ =0.002m – debljina ploče
- λ=0.23W/mK – toplinska vodljivost PA12

Koeficijent prijelaza topline za slučaj slobodne konvekcije ravne stjenke izračunava se na sljedeći način:

$$\alpha = \frac{Nu \cdot \lambda}{H} \tag{63}$$

$$Nu = 0.48\sqrt[4]{\frac{T_s - T_o}{T_o} \cdot \frac{gH^3}{\nu_s^2}} \tag{64}$$

- H=0.09 m – visina ploče
- λ=0.027 W/(mK) – toplinska vodljivost zraka pri temperaturi od 40°C
- T_s=328 K – temperatura stjenke
- T_o=313 K – temperatura okoline
- ν=16.92·10^{-6} m^2/s – kinematička viskoznost zraka pri 40°C

Za okolinu je uzeta temperatura 40°C jer se kućište ne nalazi u blizini motora niti drugih izvora topline. Temperatura stjenke se mora pretpostaviti, kasnije će se provjeriti da li je pretpostavka ispravna.

$$Nu = 0.48\sqrt[4]{\frac{328-313}{313}\cdot\frac{9.81\cdot 0.09^3}{(16.92\cdot 10^{-6})^2}} = 15.88 \quad \textbf{(65)}$$

$$\alpha_v = \frac{15.88\cdot 0.027}{0.09} = 4.76\frac{\mathrm{W}}{\mathrm{m^2K}} \quad \textbf{(66)}$$

Temperatura vanjske stjenke kućišta može se izračunati na sljedeći način:

$$\Phi = A\cdot\alpha_v\cdot(\vartheta_s - \vartheta_o) \quad \textbf{(67)}$$

$$\vartheta_s = \frac{\Phi}{A\cdot\alpha_v} + \vartheta_o = \frac{2.5}{0.035\cdot 4.76} + 313 = 328K \quad \textbf{(68)}$$

- A=0.035 m^2 – površina kućišta iz modela(bez prednje ploče)

Može se vidjeti da je temperatura stjenke dobro pretpostavljena. Pretpostavit će se da je koeficijent prijelaza topline s unutarnje strane jednak kao i s vanjske s obzirom da je visina stjenke ista, jedino temperatura zraka nešto viša. Koeficijent prolaza topline iz (62) jednak je:

$$k = \frac{1}{\frac{1}{4.76} + \frac{0.002}{0.23} + \frac{1}{4.76}} = 2.33 \frac{\mathrm{W}}{\mathrm{m^2K}} \quad \textbf{(69)}$$

Temperatura u unutrašnjosti kućišta iznosi:

$$\vartheta_{\infty u} = \frac{\Phi}{kA} + \vartheta_{\infty v} = \frac{2.5}{2.33 \cdot 0.035} + 313 = 343K = 70°C \quad \textbf{(70)}$$

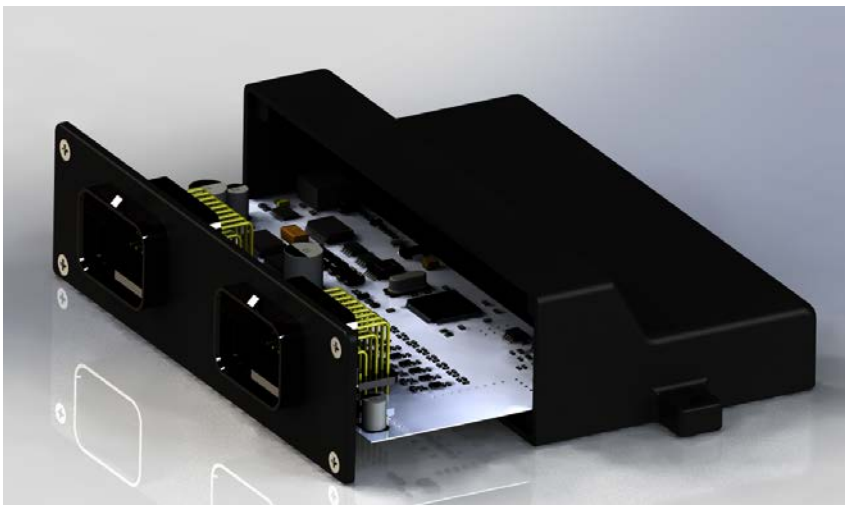
Iskustveni podatak prema [9] kaže da je kod slobodne konvekcije tipične tiskane pločice potrebno oko 15cm2 za disipaciju 1W snage uz povećanje temperature za 40°C.

$$\frac{15}{1} = \frac{A_{pcb}}{P} \rightarrow P = \frac{1 \cdot 15 \cdot 7}{15} = 7W \quad \textbf{(71)}$$

Kako je površina pločice 105cm^2, može se disipirati 7W uz istu promjenu temperature. Za snagu od 2.5W porast temperature jednak je:

$$\frac{7}{2.5} = \frac{40}{\vartheta_{pcb}} \rightarrow \vartheta_{pcb} = \frac{40 \cdot 2.5}{7} = 15°C \quad \textbf{(72)}$$

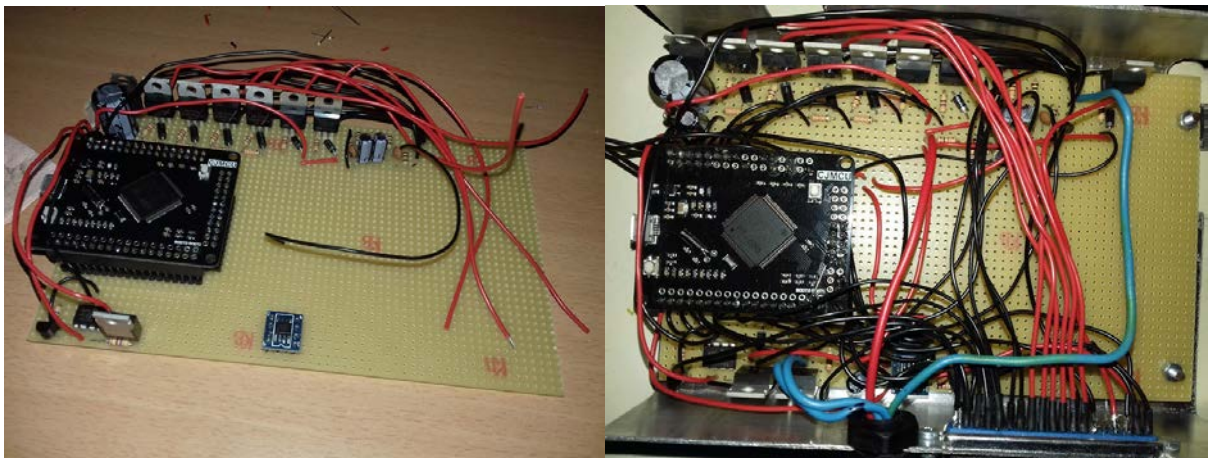
Temperatura pločice prema tome bila bi 15°C viša od temperature zraka u kućištu tj. 85°C. S obzirom da sve odabrane komponente imaju dozvoljenu temperaturu 105°C ili više može se reći da je temperatura u dozvoljenim granicama.



Slika 64. Tiskana pločica s kućištem

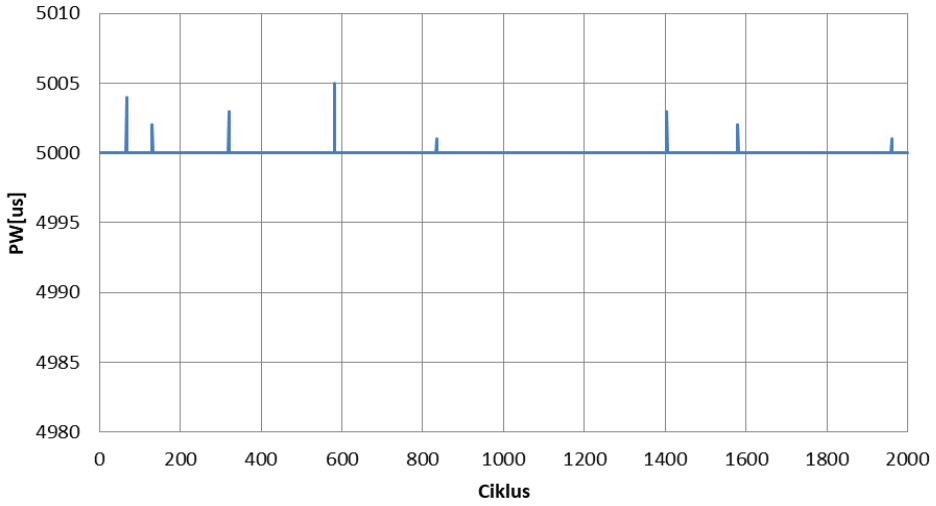
6. Testiranje sustava

Zbog nedovršenosti motora nije bilo moguće izvesti testiranje ECU-a na bolidu prije roka predaje ovog rada. Zato je prototipna verzija ECU-a, prikazana na slici 65., testirana na četverocilindričnom motoru iz Honde Civic.



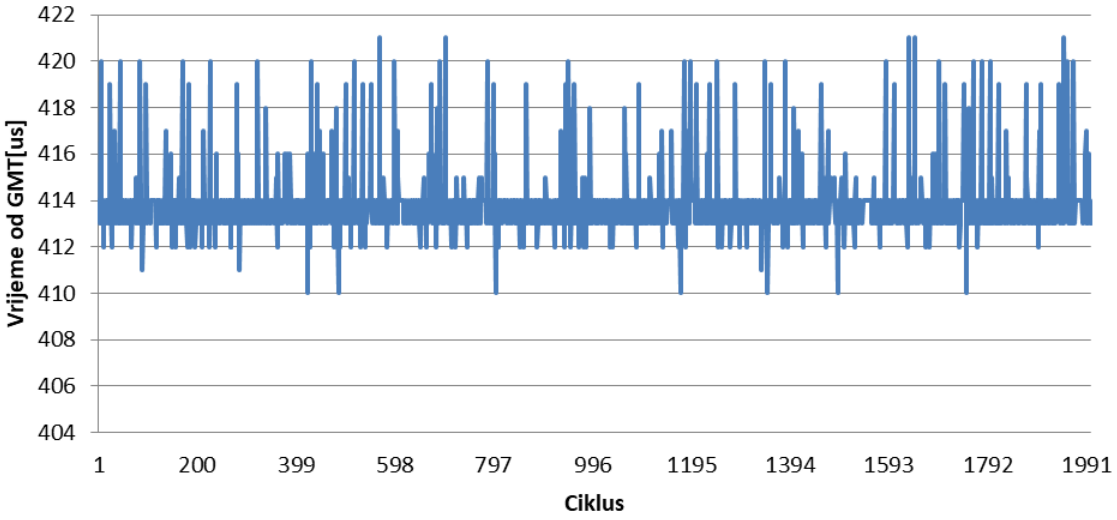
Slika 65. Eksperimentalna verzija ECU-a

Glavni cilj bio je testirati upravljački program pa je *hardware* pojednostavljen zbog lakše izrade eksperimentalne pločice. Ključne komponente kao što su mikrokontroler, NCV1124 za kondicioniranje signala induktivnog senzora, EEPROM i TC4427 *driver* za bobine korištene su iste kao i u konačnoj verziji. Pojednostavljeno je napajanje i zaštite ulaznih kanala jer ne utječu na rad samog ECU-a. Prije testiranja na motoru, ispitana je ispravnost programa simuliranjem signala senzora radilice pomoću drugog mikrokontrolera i ostalih analognih ulaza pomoću potenciometra. Nakon što je provjerena ispravnost algoritama računanja duljine otvorenosti brizgaljke i kuta pretpaljenja prema trenutno zadanoj radnoj točki krenulo se na provjeravanje točnosti otvaranja i zatvaranja brizgaljke. Brojačem s rezolucijom od jedne mikrosekunde mjerilo se vrijeme od trenutka kada je zadano otvaranje brizgaljke do trenutka zatvaranja brizgaljke. Vremena su ispisivana preko serijskog porta na računalo i uspoređivana sa zadanom duljinom otvorenosti brizgaljke. Mjerenja su provedena za nekoliko radnih točaka.



Slika 66. Mjerena duljina ubrizgavanja kroz 2000 ciklusa

U prikazanom primjeru zadana duljina ubrizgavanja je *PW*=5000µs .Može se vidjeti da su odstupanja na razini nekoliko mikrosekundi što nema nikakvog značajnog utjecaja na količinu ubrizganog goriva s obzirom da je brizgaljka u praznom hodu otvorena oko 2ms. Do kašnjenja zatvaranja dolazi zbog toga što prekidne funkcije u kojima se zadaje otvaranje i zatvaranje brizgaljke imaju manji prioritet nego one za čitanje senzora koljenastog vratila. Što znači da ako se trenutno izvršava druga prekidna funkcija, brizgaljka se neće moći zatvoriti dok ona ne završi. Na sličan način napravljena je i provjera početka punjenja i pražnjenja bobine, samo što je vrijeme mjereno kao udaljenost od GMT. Ako se zada neki kut pretpaljenja, a brzina vrtnje je poznata može se izračunati vremenska udaljenost trenutka paljenja od GMT. Izračunata vremenska udaljenost uspoređuje se s izmjerenom.



Slika 67. Mjerena udaljenost trenutka paljenja od GMT kroz 2000 ciklusa

U prikazanom primjeru zadani kut pretpaljenja je 20° prije GMT pri brzini vrtnje 8000min^{-1}. Za jednu mikrosekundu, koljenasto vratilo pri brzini vrtnje od 8000min-1 napravi kut od 0.05°:

$$\varphi = \omega \cdot t = \frac{8000}{60} \cdot 360 \cdot 10^{-6} = 0.048° \quad \textbf{(73)}$$

- ω – kutna brzina [°/s]
- t – vrijeme [s]

Što bi značilo da kašnjenje od 6μs dovodi do pomicanja kuta pretpaljenja bliže GMT za 0.29°. Nakon što je ispitana ispravnost osnovnih funkcija sustav je ugrađen u vozilo.

7. Zaključak

Cilj ovog rada bio je napraviti upravljačku jedinicu motora bolida formule student i pružiti korisne informacije vezane uz *hardware* i samo upravljanje radom motora. S obzirom da je ovo prvi pothvat u izradi upravljačke jedinice motora u projektu Formula Student FSB Racing Teama, želja je bila napraviti dobru podlogu za daljnji razvoj.

Razvoj ovakvog proizvoda vrlo je zahtjevan jer zahtjeva dobro poznavanje elektronike, izrade shema, projektiranja tiskanih pločica, programiranja mikrokontrolera i rada motora s unutarnjim izgaranjem. U jednom radu ovog opsega nije moguće detaljno prikazati i razraditi svu problematiku koja se veže uz ovakav projekt. Prikazane su osnove principa rada korištenih senzora i aktuatora, strujni krugovi za kondicioniranje signala i upravljanje aktuatorima te zadatci koje upravljačka jedinica mora izvršavati. Objašnjen je princip rada upravljačkog programa i dana su tri modela prema kojima se može određivati potrebna količina ubrizganog goriva ovisno o senzoru koji se koristi za mjerenje opterećenja. Trenutno je u programu implementirana *Speed density* metoda koja procjenjuje maseni protok zraka u motor prema brzini vrtnje, tlaku u usisnom kolektoru i volumetrijskoj efikasnosti definiranoj od strane korisnika. Pokazane su pojave dinamike goriva u usisnoj cijevi kod nestacionarnih uvjeta rada i zagrijavanja motora te potrebne korekcije ubrizgavanja kako bi se osigurao ispravan rad u svim uvjetima. Može se reći da održavanje zadanog faktora pretička zraka u nestacionarnim uvjetima rada tj. kod naglih promjena kuta zaklopke predstavlja najveći problem u upravljanju radom Ottovog motora i tu ima prostora za poboljšanje. Dvije metode određivanja korekcije ubrizgavanja kod nestacionarnih uvjeta su prikazane, jedna se temelji na očitavanju *lookup* tablica prema trenutnim radnim uvjetima i brzini promjene kuta zaklopke, a druga na X-Tau modelu koji prati kondenziranje i isparavanje goriva na stjenkama u usisnoj cijevi. Obje metode zahtijevaju dugotrajno podešavanje ako se želi postići održavanje smjese u uskom rasponu kod svih radnih točaka.

Zbog vrlo male generirane topline komponenata, odlučeno je da će se kućište izraditi 3D printanjem iz poliamida PA12 čime se postiže mala masa i omogućuje se izrada složenijih oblika. Jedna od prednosti vlastitog sustava je mogućnost izmjene upravljačkog koda i implementacije naprednijih modela upravljanja u budućnosti.

LITERATURA

[1] Horst Bauer; Bosch: Gasoline-engine management, Robert Bosch GmbH, Stuttgart, 1999.

[2] Tehnika motornih vozila, Pučko otvoreno učilište, Zagreb, 2006

[3] Ivan Mahalec, Zoran Lulić, Darko Kozarac: Motori s unutarnjim izgaranjem, Fakultet strojarstva i brodogradnje, Zagreb, 2013.

[4] Konrad Reif; Bosch : Diesel Engine Management, Springer, Wiesbaden, 2014

[5] http://www.megamanual.com/ms2/

[6] http://www.wbo2.com/

[7] Fairchild semiconductors : Introduction to automotive ignition systems, Aurora, 2014

[8] Di Gaeta, Alessandro & Glielmo, Luigi & Santini, Stefania & Cioffi, Vittorio & di Giuseppe, Carlo, On-Line Adaptation of Wall-Wetting Model Parameters, 2001

[9] Texas Instruments aplication report SNVA419C, 2013

[10] Aquino, C., "Transient A/F Control Characteristics of the 5 Liter Central Fuel Injection Engine," SAE Technical Paper 810494, 1981

[11] Shayler, P., Teo, Y., and Scarisbrick, A., "Fuel Transport Characteristics of Spark Ignition Engines for Transient Fuel Compensation," SAE Technical Paper 950067, 1995

[12] Greg Banish: Engine Management Advanced Tuning, CarTech, Chicago, 2007

[13] Jeff Hartman: Automotive Engine Management Systems, MBI, Minneapolis, 2013

[14] Bosch: Technical product information, Planar Wide Band Lambda sensor, 2005

[15] NXP Technical Data: Automotive Engine Control IC MC33810, 2014

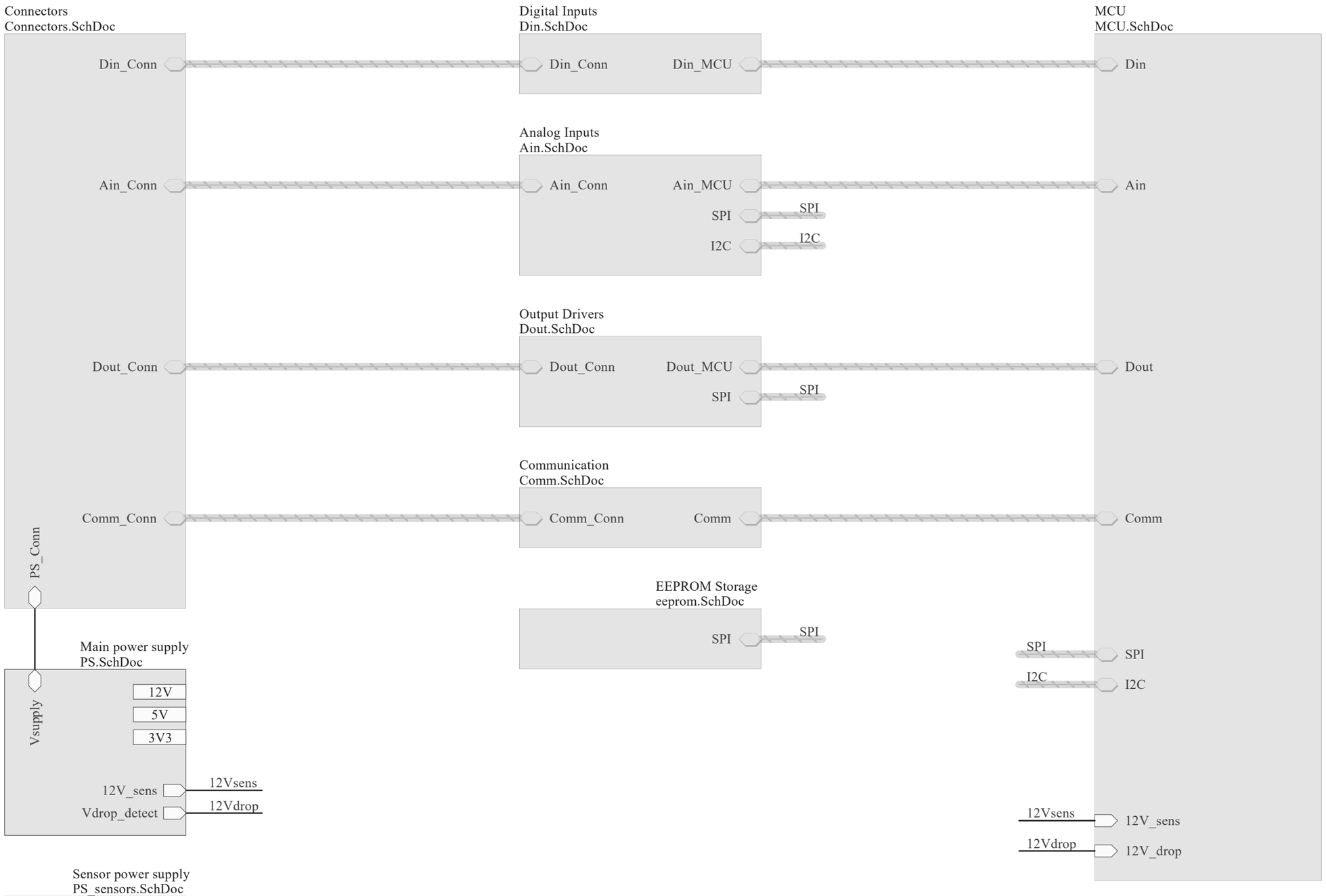
[16] STMicroelectronics Technical Data: Automotive-grade dual N-channel MOSFET STS8DN6LF6AG

[17] Boris Halasz: Predlošci za vježbe iz kolegija Uvod u termodinamiku, Fakultet strojarstva i brodogradnje, Zagreb, 2011

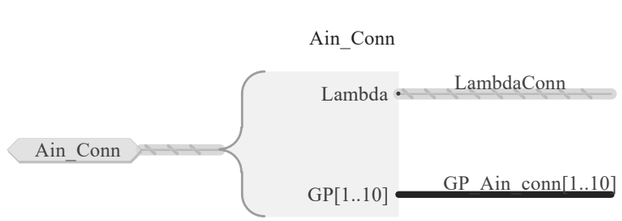
PRILOZI

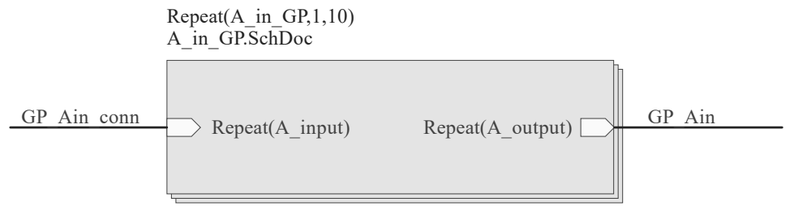
I. CD-R disc(Sheme, programski kod)

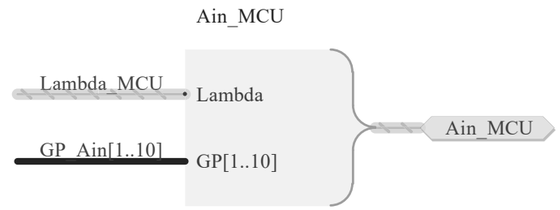
II. Tehnička dokumentacija

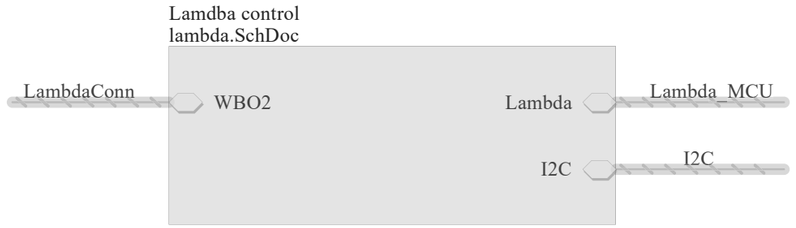


Naziv: Glavni list			Projektirao: Nick Findrik Marcel Besedić Mateo Milovac	FSB Zagreb
Format: A3	Number:ECU-0-0	Revizija:		
Datum: 21.3.2018.		List 1 od 18	Pregledao:	
Dokument TopSheet.SchDoc				

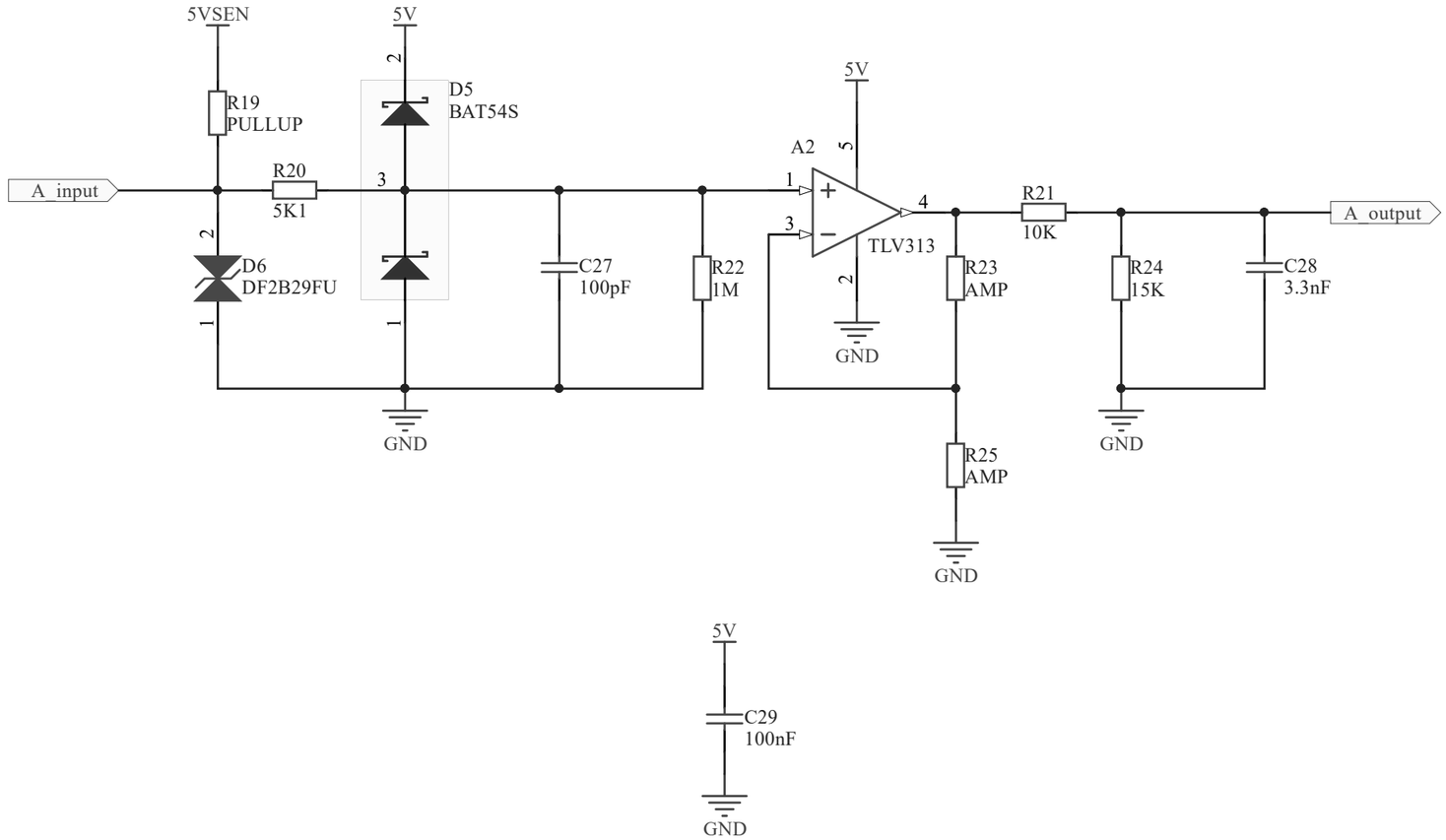




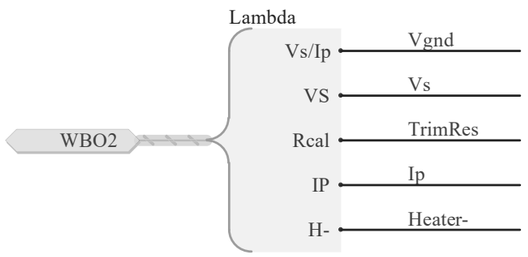


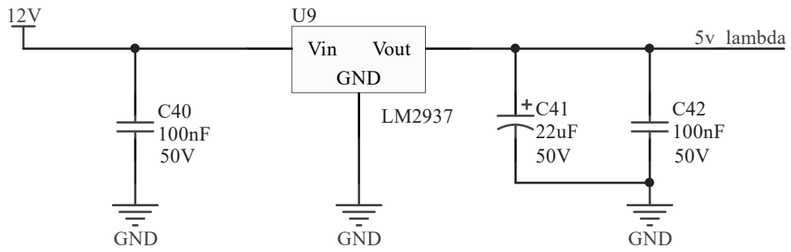


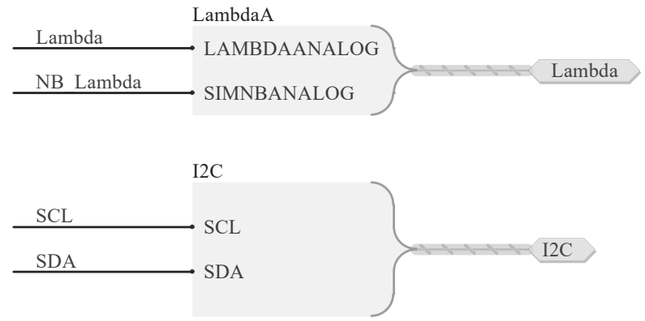
Naziv: Analogni ulazi			Projektirao: Nick Findrik Marcel Besedić Mateo Milovac	FSB Zagreb
Format: A3	Number:ECU-1-0	Revizija:		
Datum: 21.3.2018.		List 2 od 18	Pregledao:	
Dokument Ain.SchDoc				

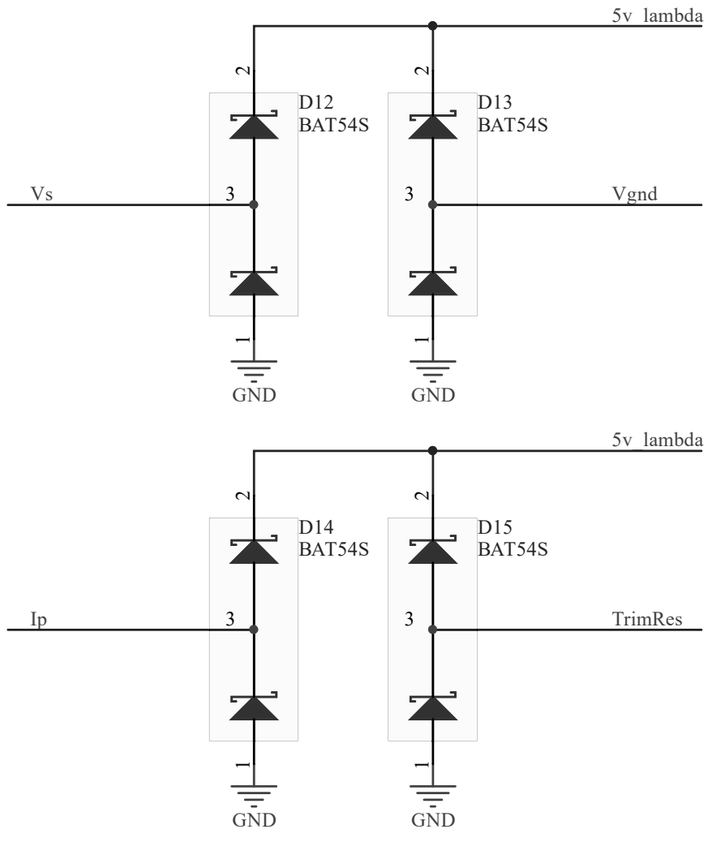


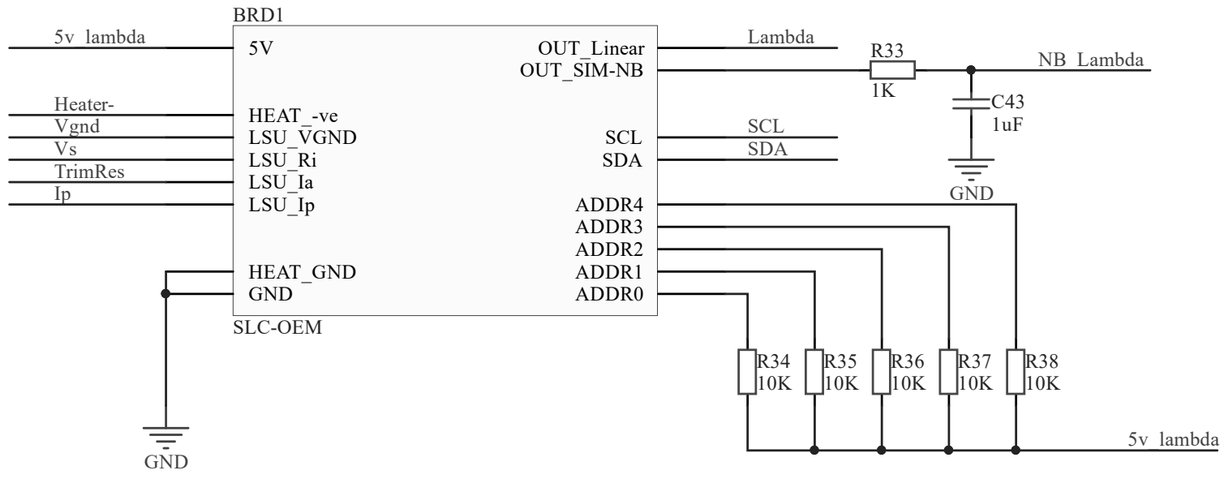
Naziv: Analogni ulaz GP			Projektirao: Nick Findrik, Marcel Besedić, Mateo Milovac
Format: A3	Number: ECU-1-1	Revizija:	Pregledao:
Datum: 21.3.2018.		List 3 od 18	FSB Zagreb
Dokument A_in_GP.SchDoc			



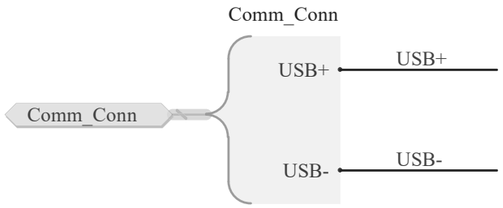


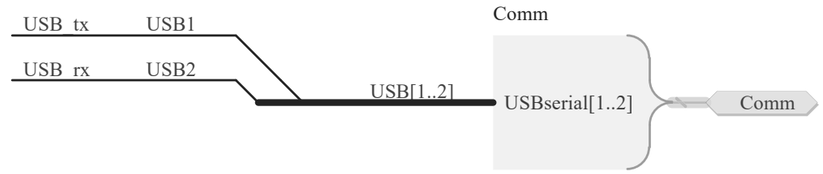


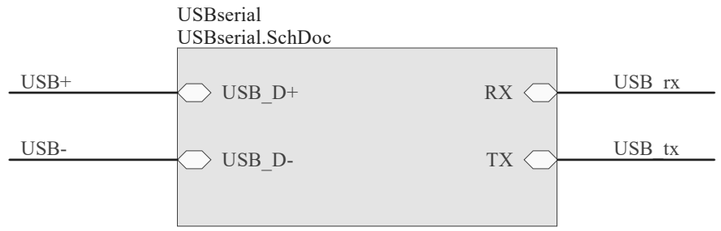




Naziv: WBO2 kontroler			Projektirao: Nick Findrik Marcel Besedić Mateo Milovac	FSB Zagreb
Format: A3	Number:ECU-1-2	Revizija:		
Datum: 21.3.2018.		List 4 od 18	Pregledao:	
Dokument lambda.SchDoc				

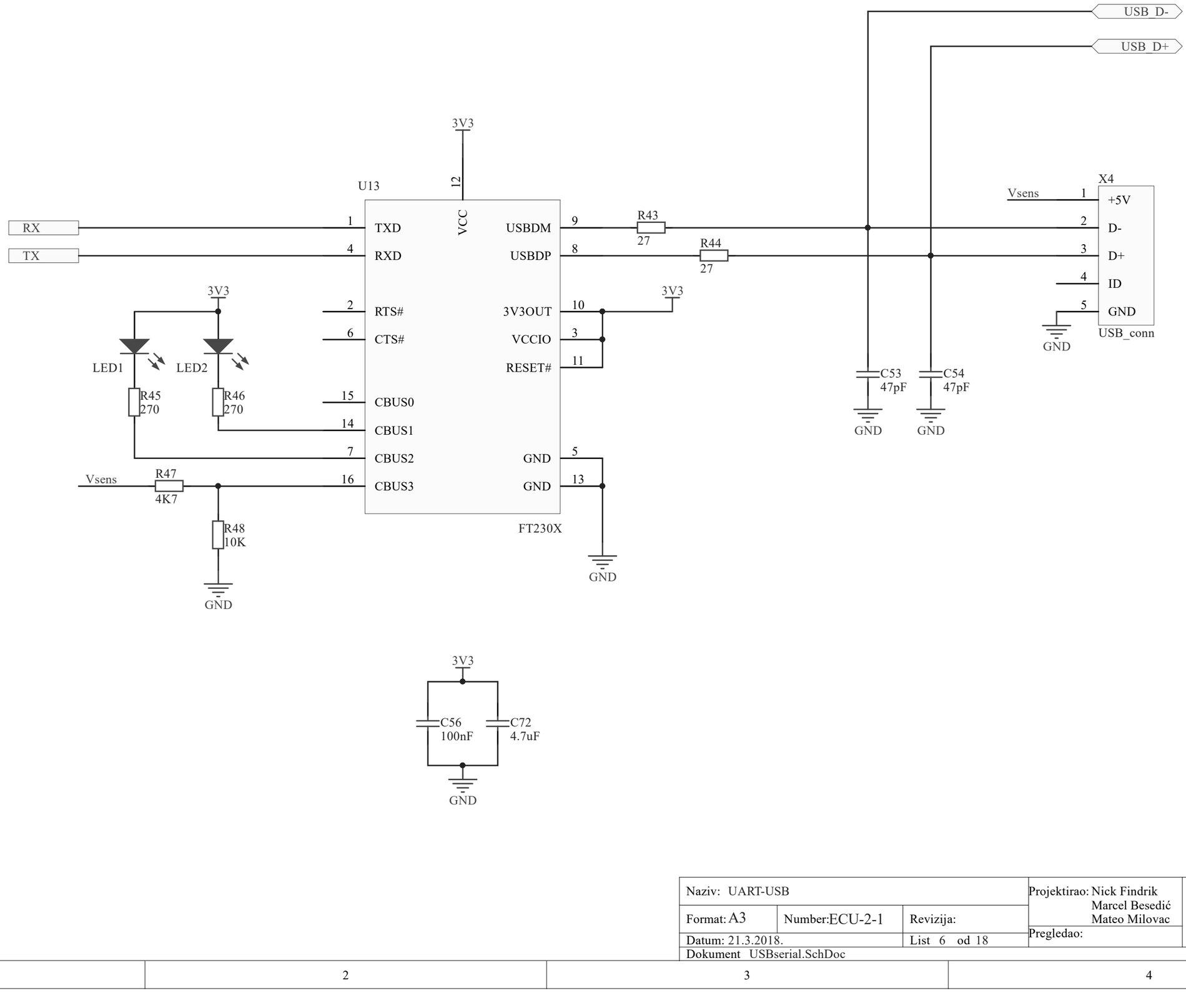




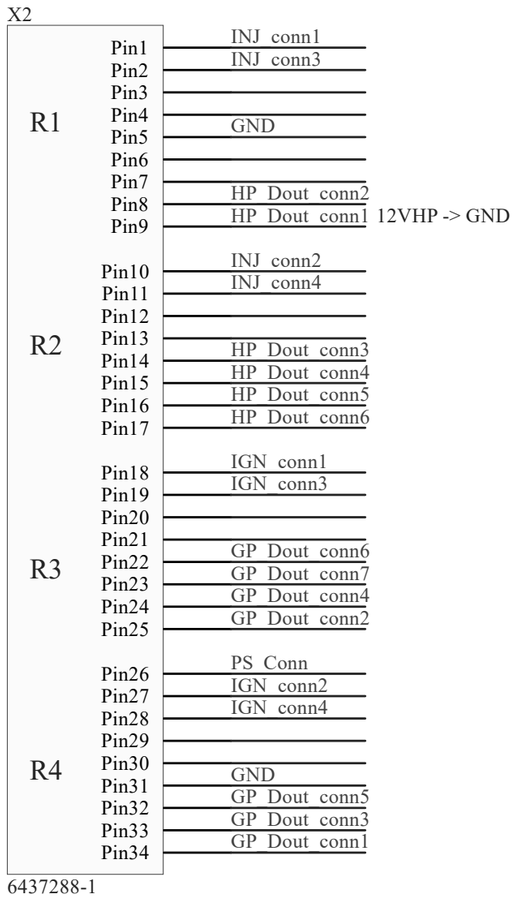


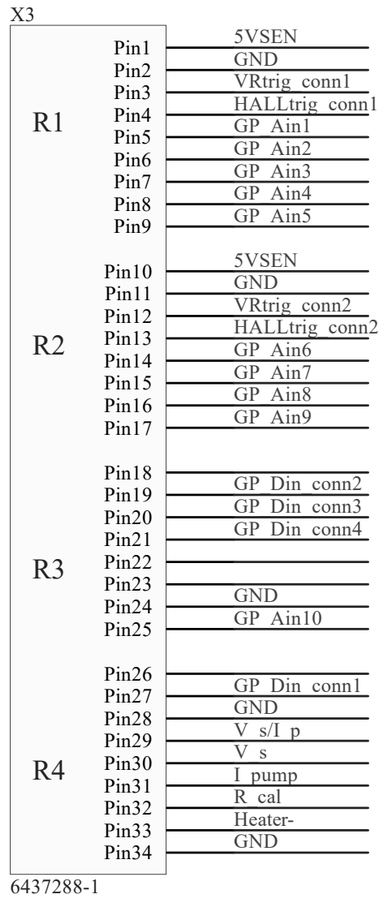
Naziv: Komunikacija			Projektirao: Nick Findrik, Marcel Besedić, Mateo Milovac
Format: A3	Number: ECU-2-0	Revizija:	
Datum: 21.3.2018.		List 5 od 18	Pregledao:
Dokument Comm.SchDoc			

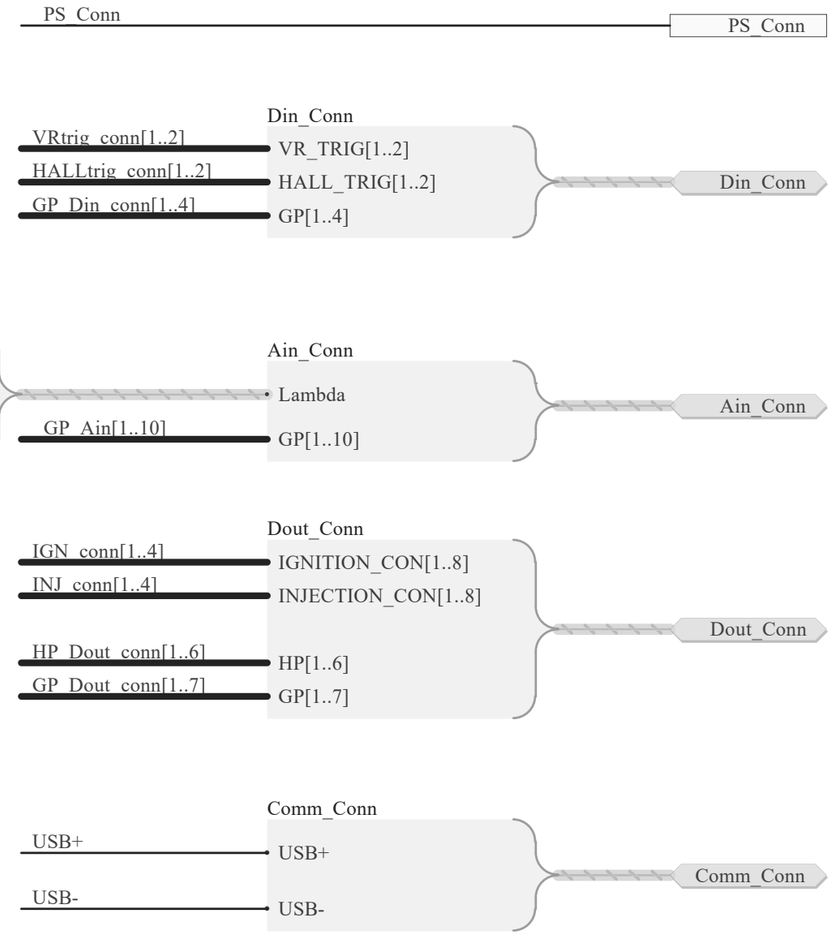
FSB Zagreb

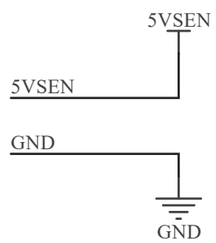


Naziv: UART-USB		Projektirao: Nick Findrik, Marcel Besedić, Mateo Milovac
Format: A3	Number: ECU-2-1	Revizija:
Datum: 21.3.2018.	List 6 od 18	Pregledao:
Dokument USBserial.SchDoc		FSB Zagreb

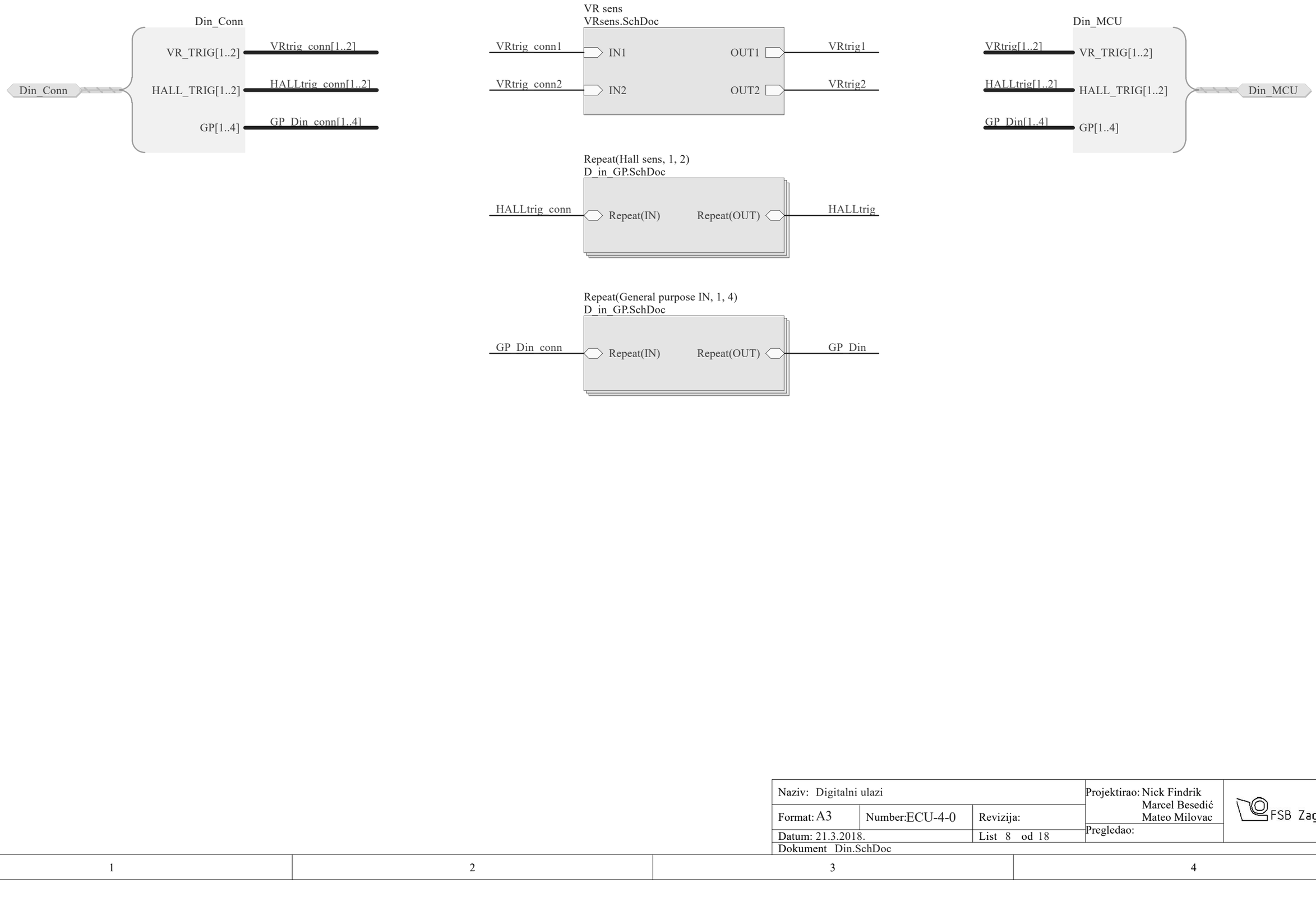






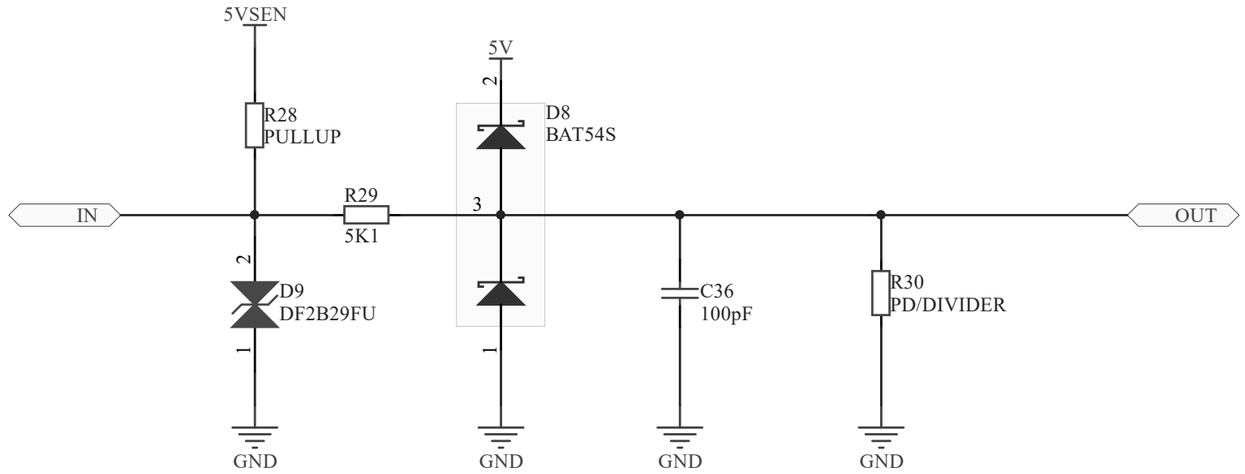


Naziv: Konektori			Projektirao: Nick Findrik, Marcel Besedić, Mateo Milovac
Format: A3	Number: ECU-3-0	Revizija:	
Datum: 21.3.2018.		List 7 od 18	Pregledao:
Dokument Connectors.SchDoc			FSB Zagreb

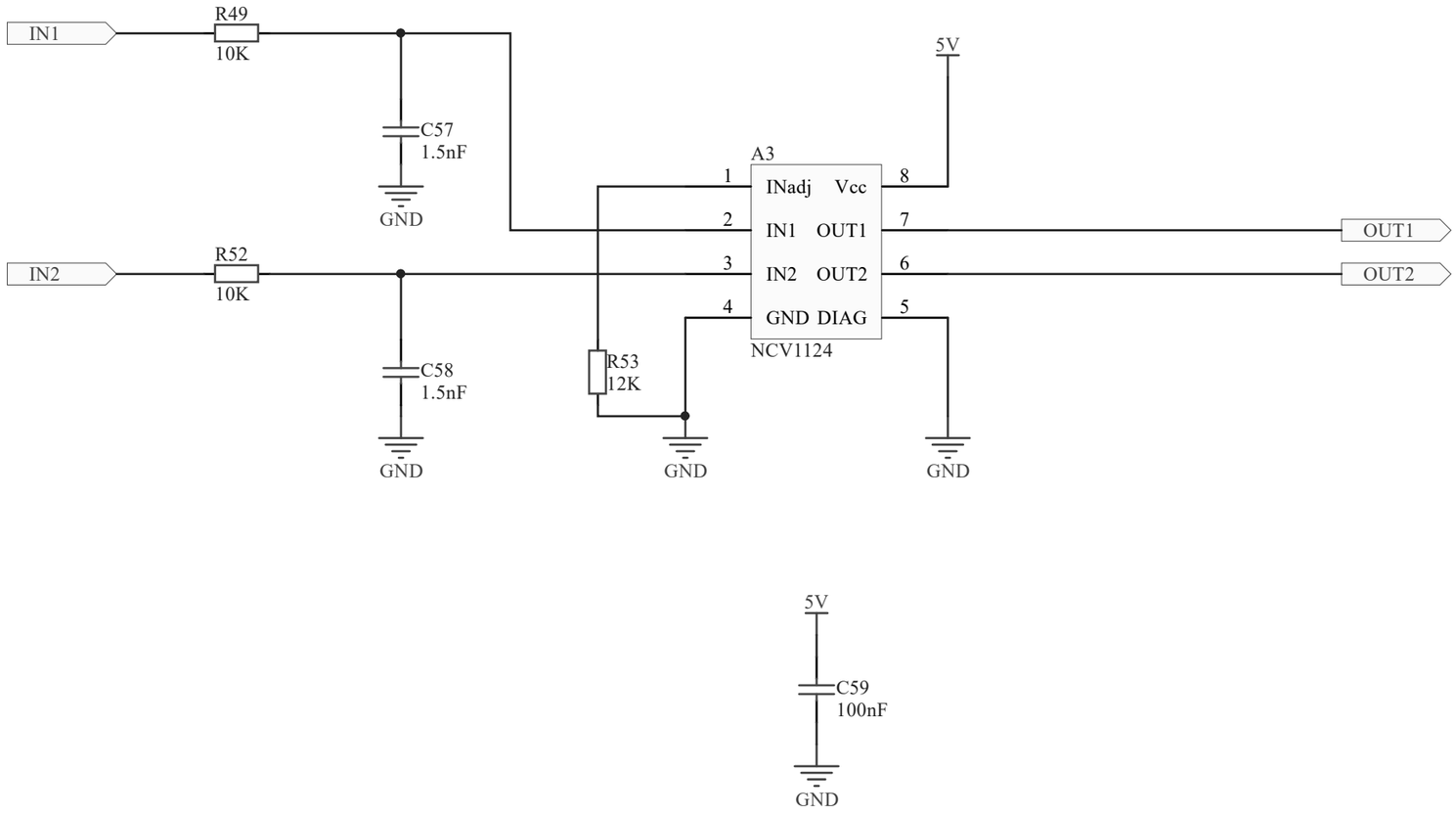


Naziv: Digitalni ulazi			Projektirao: Nick Findrik Marcel Besedić Mateo Milovac
Format: A3	Number:ECU-4-0	Revizija:	
Datum: 21.3.2018.		List 8 od 18	Pregledao:
Dokument Din.SchDoc			

FSB Zagreb

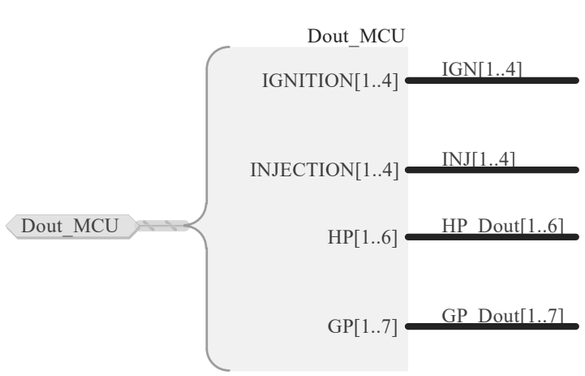


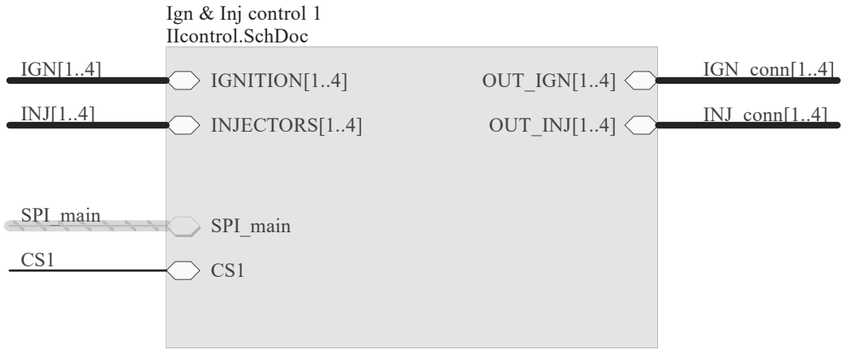
Naziv: Digitalni ulaz GP			Projektirao: Nick Findrik Marcel Besedić Mateo Milovac	FSB Zagreb
Format: A3	Number:ECU-4-1	Revizija:		
Datum: 21.3.2018.		List 9 od 18	Pregledao:	
Dokument D_in_GP.SchDoc				

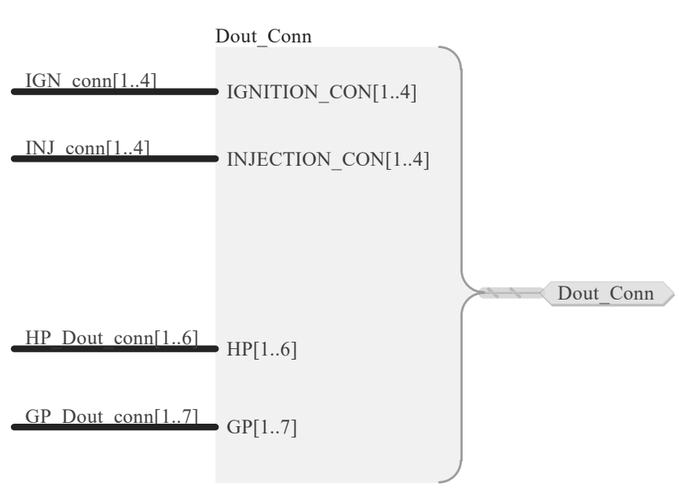


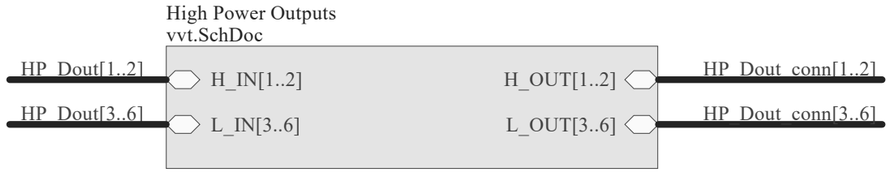
Naziv: VR conditioner			Projektirao: Nick Findrik Marcel Besedić Mateo Milovac
Format: A3	Number:ECU-4-2	Revizija:	
Datum: 21.3.2018.		List 10 od 18	Pregledao:
Dokument VRsens.SchDoc			

FSB Zagreb

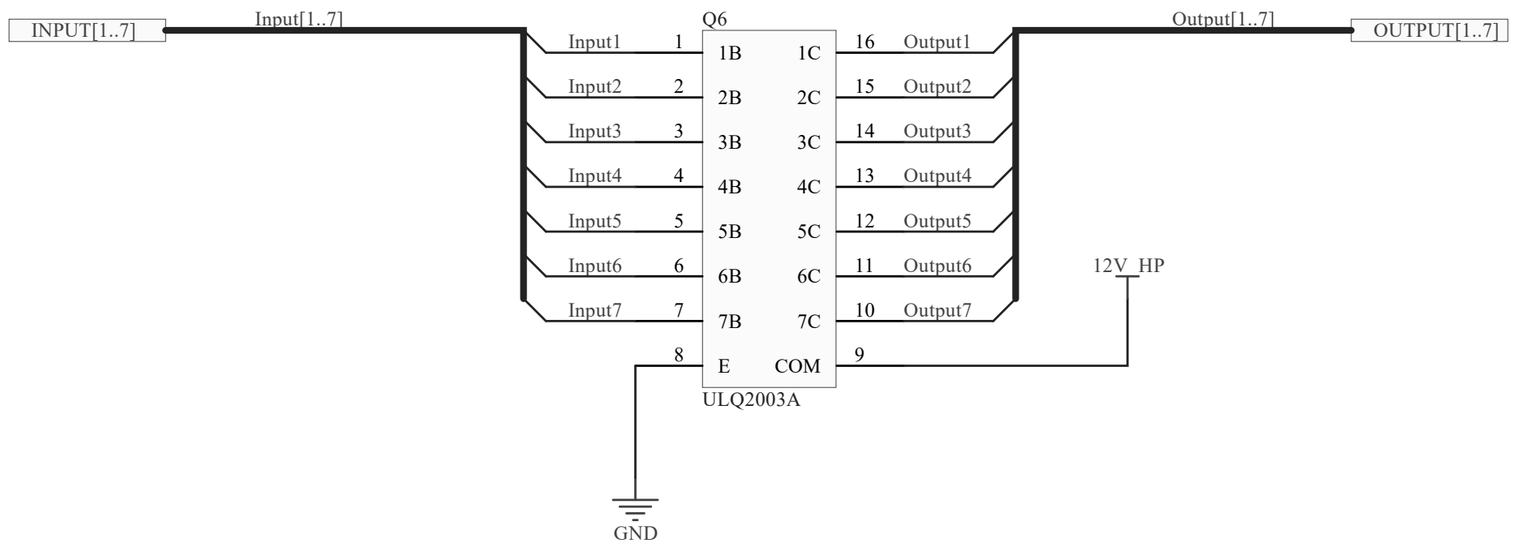




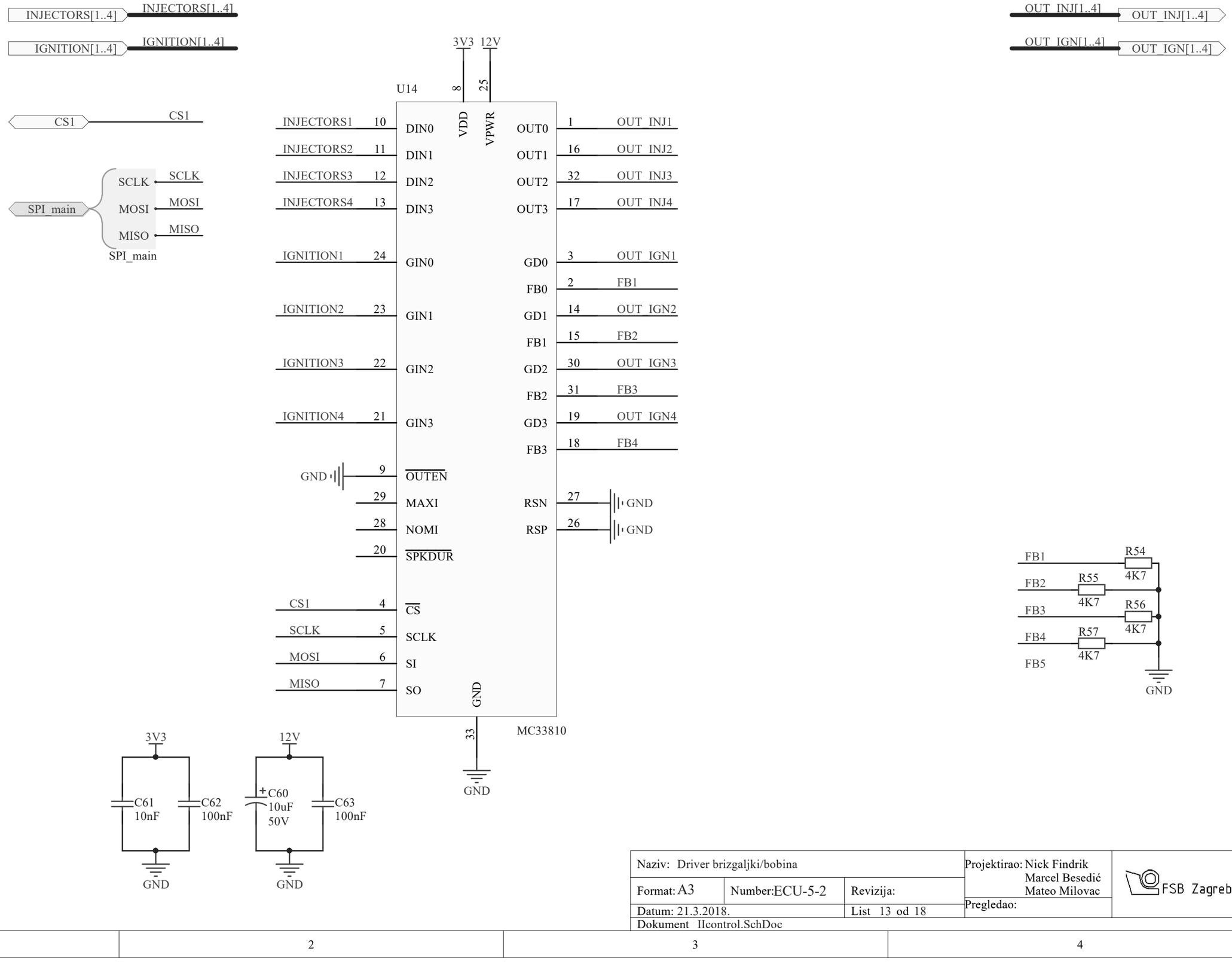




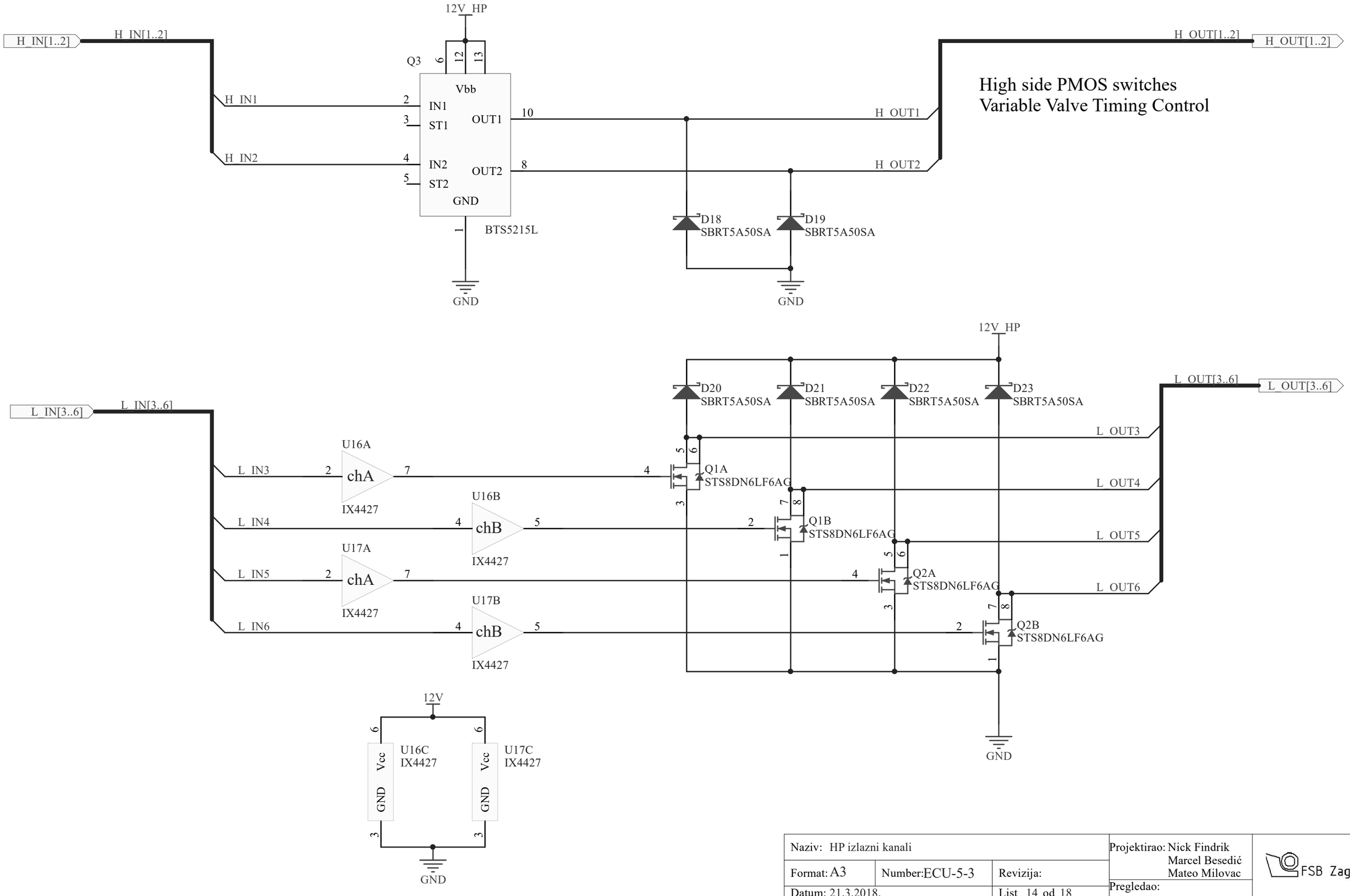
Naziv: Izlazni kanali			Projektirao: Nick Findrik Marcel Besedić Mateo Milovac	FSB Zagreb
Format: A3	Number: ECU-5-0	Revizija:		
Datum: 21.3.2018.		List 11 od 18	Pregledao:	
Dokument Dout.SchDoc				



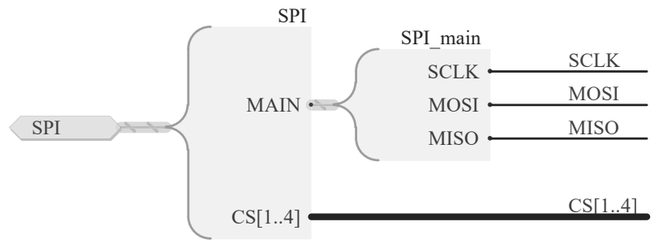
Naziv: Relay driver			Projektirao: Nick Findrik, Marcel Besedić, Mateo Milovac
Format: A3	Number: ECU-5-1	Revizija: *	Pregledao:
Datum: 21.3.2018.		List 12 od 18	FSB Zagreb
Dokument general.SchDoc			

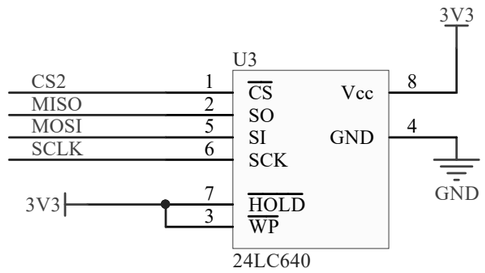


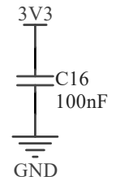
Naziv: Driver brizgaljki/bobina		Projektirao: Nick Findrik, Marcel Besedić, Mateo Milovac
Format: A3	Number: ECU-5-2	Revizija:
Datum: 21.3.2018.	List 13 od 18	Pregledao:
Dokument IIcontrol.SchDoc		FSB Zagreb



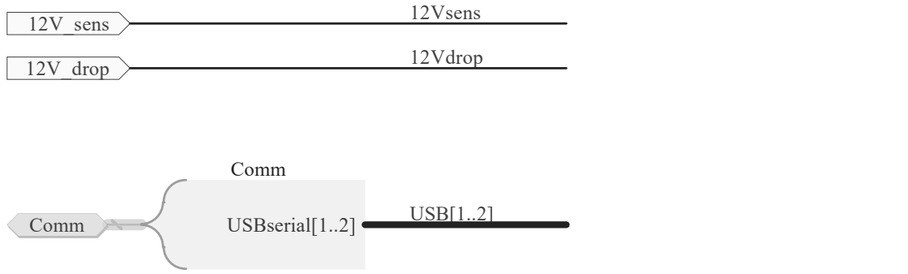
Naziv: HP izlazni kanali			Projektirao: Nick Findrik, Marcel Besedić, Mateo Milovac
Format: A3	Number: ECU-5-3	Revizija:	Pregledao:
Datum: 21.3.2018.		List 14 od 18	FSB Zagreb
Dokument vvt.SchDoc			

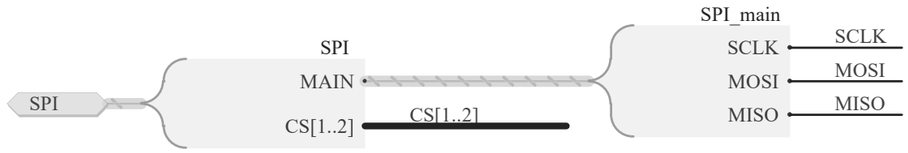


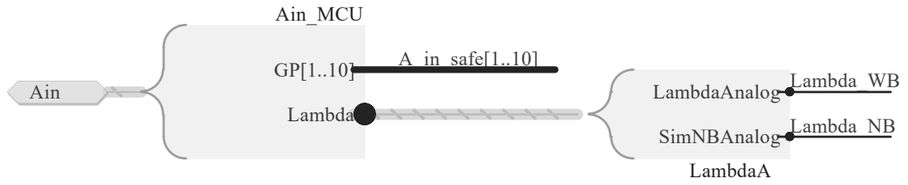


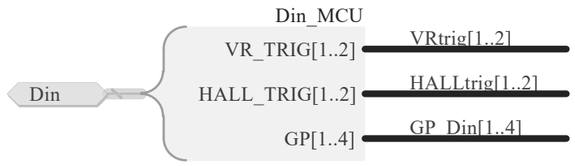


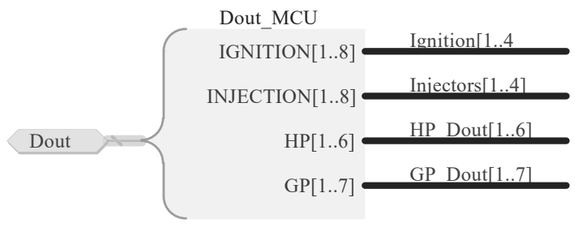
Naziv: EEPROM			Projektirao: Nick Findrik Marcel Besedić Mateo Milovac	FSB Zagreb
Format: A3	Number:ECU-6-0	Revizija: *		
Datum: 21.3.2018.		List 15 od 18	Pregledao:	
Dokument EEPROM.SchDoc				

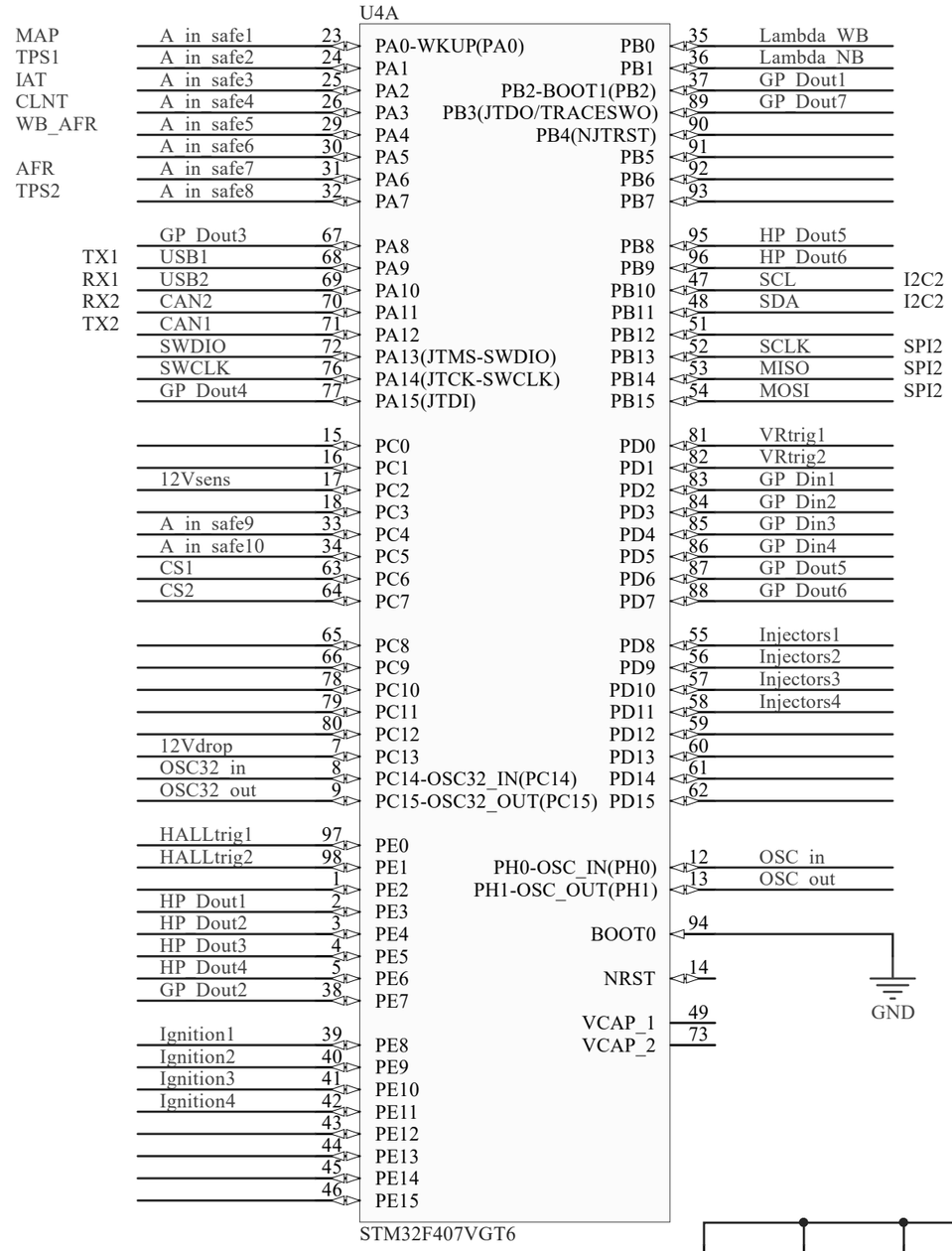


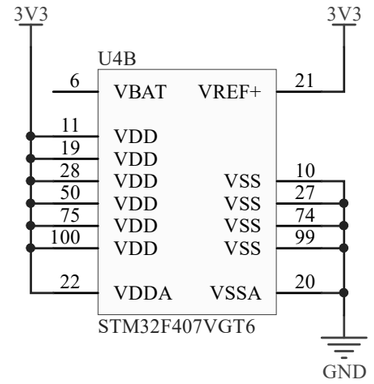


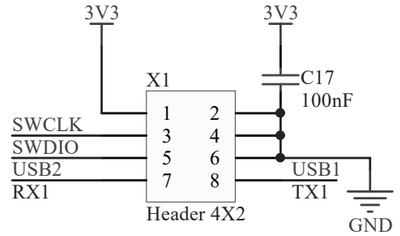


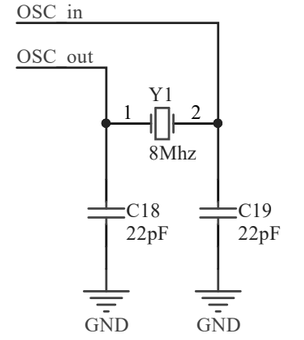


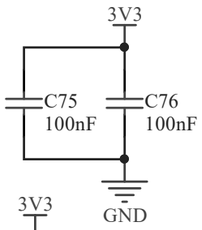


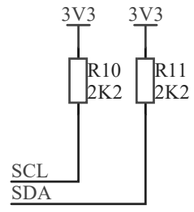


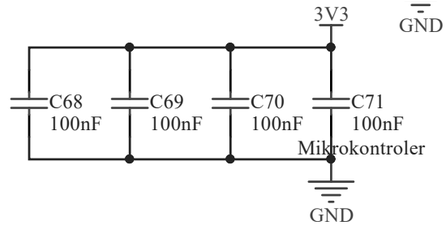


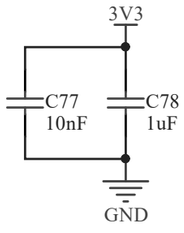




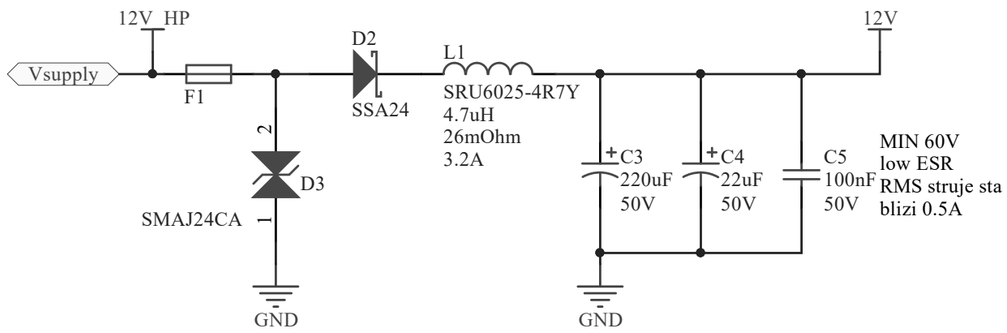


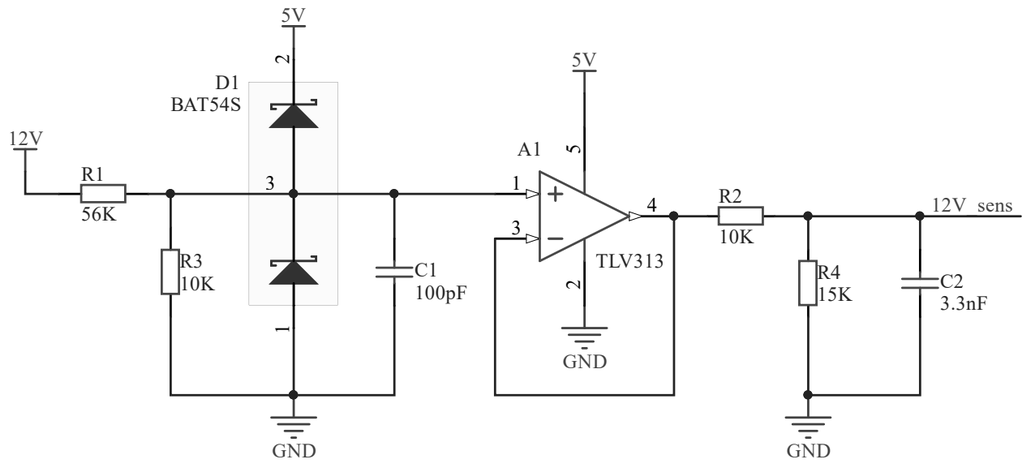


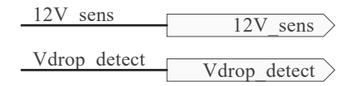


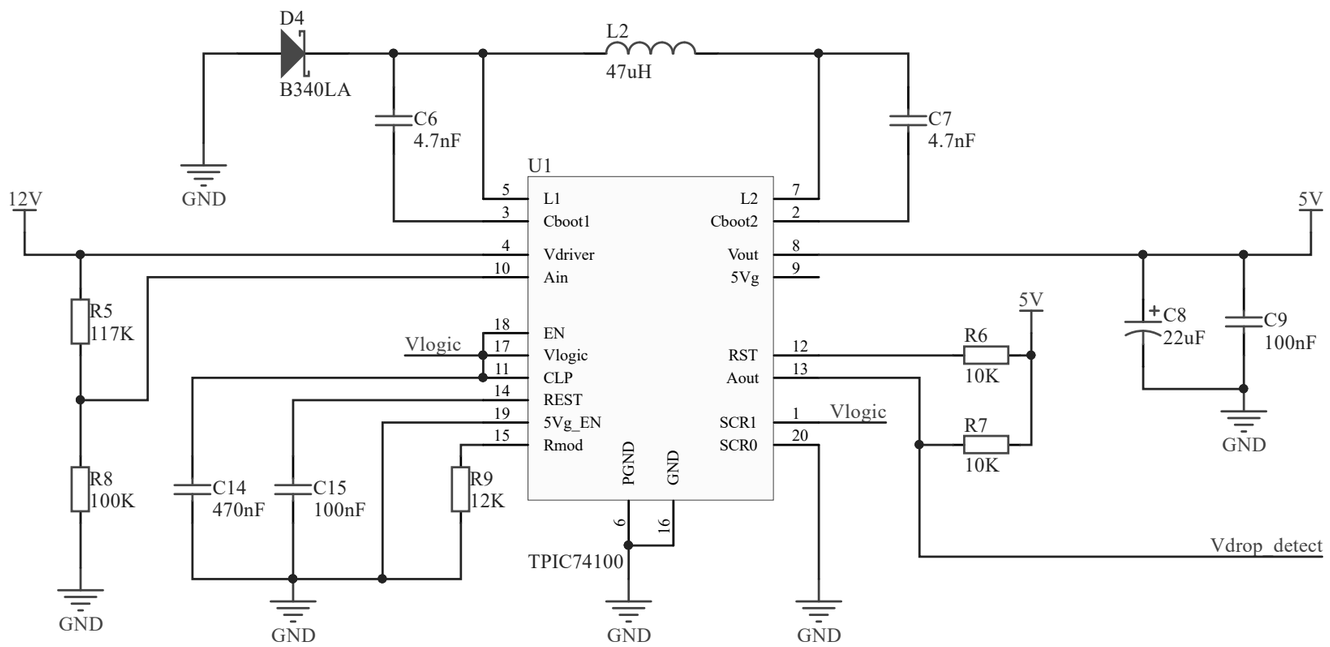


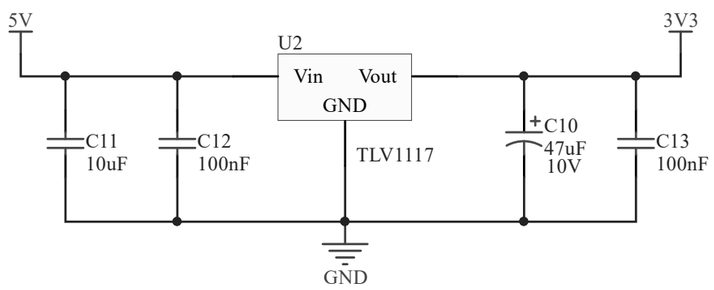
Naziv:			Projektirao: Nick Findrik, Marcel Besedić, Mateo Milovac
Format: A3	Number: ECU-7-0	Revizija:	
Datum: 21.3.2018.		List 16 od 18	Pregledao:
Dokument MCU.SchDoc			FSB Zagreb



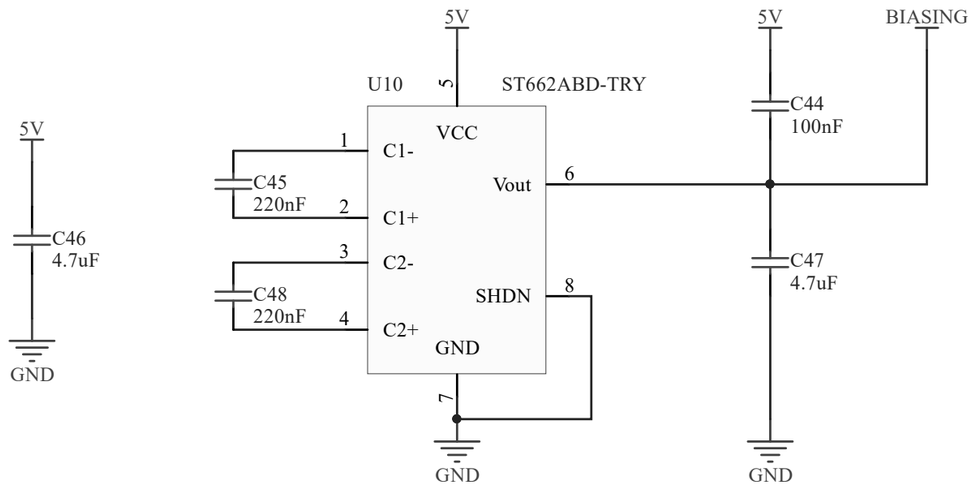


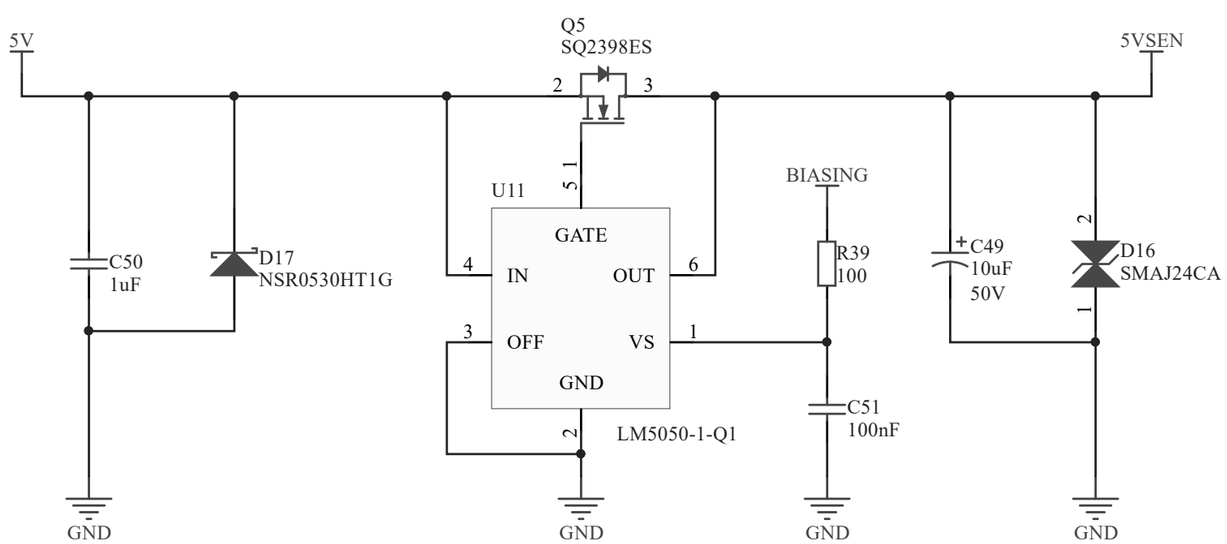






Naziv: Regulatori napona			Projektirao: Nick Findrik Marcel Besedić Mateo Milovac	FSB Zagreb
Format: A3	Number:ECU-8-0	Revizija:		
Datum: 21.3.2018.		List 17 od 18	Pregledao:	
Dokument PS.SchDoc				





Naziv: Zaštita vanjskih 5V			Projektirao:	Nick Findrik Marcel Besedić Mateo Milovac
Format: A3	Number: ECU-9-0	Revizija:		
Datum: 21.3.2018.		List 18 od 18	Pregledao:	
Dokument PS_sensors.SchDoc				FSB Zagreb

```
#include <libmaple/scb.h>
#include <SPI.h>

#define SPI2_NSS_PIN PB12   //SPI_2 Chip Select pin is PB12
SPIClass SPI_2(2); //Create an instance of the SPI Class called SPI_2 that uses the 2nd SPI
Port


byte f1[18][18] =       {

{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  128,  130,  135,  135,  135,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  125,  125,  130,  130,  130,  125,  120,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  125,  125,  125,  125,  125,  120,  115,  110,  110,
110,  110,  110 },
{ 120,  120,  120,  120,  120,  120,  120,  125,  125,  125,  125,  120,  115,  110,  110,
110,  110,  110 },
{ 108,  108,  108,  108,  115,  115,  115,  115,  115,  115,  115,  115,  115,  110,  110,
110,  110,  110 },
{ 100,  100,  100,  104,  104,  104,  105,  107,  107,  107,  107,  107,  107,  107,  107,
107,  107,  107 },
{ 90, 90, 90, 86, 88, 88, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94, 94  },
{ 85, 85, 85, 79, 80, 75, 83, 86, 86, 86, 86, 86, 86, 86, 86, 86, 86, 86  },
{ 75, 75, 75, 73, 72, 73, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75  },
{ 68, 68, 68, 68, 65, 65, 70, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75  },
{ 64, 64, 64, 63, 63, 63, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64  },
{ 46, 46, 46, 46, 46, 51, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60  }

};

int f1Rpm[18] = { 0, 560, 1120, 1680, 2240, 2800, 3360, 3920, 4480, 5040, 5600, 6160, 6720,
7280, 9500, 9600, 9700, 9800 };
int f1Map[18] = { 14, 28, 42, 56, 70, 84, 98, 112, 126, 140, 154, 168, 182, 196, 240, 250,
260, 270 };



byte f2[18][18] =       {

{ 125,  125,  125,  125,  125,  125,  135,  136,  143,  143,  143,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  135,  136,  143,  143,  143,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  135,  136,  143,  143,  143,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  135,  136,  143,  143,  143,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  135,  136,  143,  143,  143,  130,  122,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  125,  135,  136,  143,  143,  143,  135,  127,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  130,  135,  135,  135,  138,  138,  135,  127,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  130,  133,  133,  133,  133,  133,  125,  120,  110,  110,
110,  110,  110 },
{ 125,  125,  125,  125,  125,  130,  130,  130,  130,  130,  130,  123,  115,  110,  110,
110,  110,  110 },
{ 120,  120,  120,  120,  120,  120,  120,  120,  120,  120,  120,  120,  117,  110,  110,
```

```
110,  110,  110 },
{ 108,  108,  108,  108,  117,  117,  117,  117,  117,  117,  117,  117,  117,  110,  110,
110,  110,  110 },
{ 103,  103,  103,  103,  103,  107,  107,  116,  116,  116,  116,  116,  115,  107,  107,
107,  107,  107 },
{ 90, 90, 90, 90, 90, 94, 102, 102, 102, 111, 112, 115, 110, 103, 103, 103, 103, 103  },
{ 85, 85, 85, 85, 85, 86, 90, 90, 95, 96, 100, 100, 98, 98, 98, 98, 98, 98  },
{ 75, 75, 75, 75, 75, 75, 84, 84, 85, 86, 90, 95, 88, 88, 88, 88, 88, 88  },
{ 68, 68, 68, 68, 65, 65, 75, 78, 83, 85, 87, 87, 85, 85, 85, 85, 85, 85  },
{ 64, 64, 64, 63, 63, 63, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64  },
{ 46, 46, 46, 46, 46, 51, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60  }

};

int f2Rpm[18] = { 0, 560, 1120, 1680, 2240, 2800, 3360, 3920, 4480, 5040, 5600, 6160, 6720,
7280, 9500, 9600, 9700, 9800 };
int f2Map[18] = { 14, 28, 42, 56, 70, 84, 98, 112, 126, 140, 154, 168, 182, 196, 240, 250,
260, 270 };



int d[18][18] =  {

{2, 2, 3, 10, 12, 12, 12, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
{2, 2, 3, 10, 12, 12, 12, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
{2, 2, 3, 10, 12, 12, 12, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
{2, 2, 3, 10, 12, 12, 12, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
{2, 2, 3, 10, 12, 12, 12, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14 },
{2, 2, 3, 10, 13, 13, 15, 18, 18, 18, 18, 18, 20, 20, 20, 20, 20, 20 },
{2, 2, 3, 10, 13, 14, 17, 18, 18, 18, 18, 19, 22, 22, 22, 22, 22, 22 },
{2, 2, 3, 10, 13, 14, 18, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20 },
{2, 2, 3, 10, 13, 15, 20, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22 },
{10,  10, 10, 10, 14, 18, 22, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24 },
{10,  10, 10, 10, 17, 20, 24, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26 },
{10,  10, 14, 16, 19, 23, 26, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27 },
{10,  12, 15, 20, 24, 27, 29, 30, 30,  30, 30, 30, 1,  1,  1,  1,  1,  1 },
{10,  13, 17, 22, 27, 32, 32, 34, 34,  34, 34, 34, 1,  1,  1,  1,  1,  1 },
{10,  14, 18, 25, 31, 32, 34, 36, 36,  36, 36, 36, 1,  1,  1,  1,  1,  1 },
{10,  15, 20, 28, 30, 34, 37, 37, 37,  37, 37, 37, 1,  1,  1,  1,  1,  1 },
{13,  12, 12, 18, 30, 36, 40, 40, 40,  40, 40, 40, 1,  1,  1,  1,  1,  1 },
{13,  12, 12, 18, 30, 36, 40, 40, 40,  40, 40, 40, 40, 40, 40,  1,  1,  1 }

};

int dRpm[18] = { 0, 560, 1120, 1680, 2240, 2800, 3360, 3920, 4480, 5040, 5600, 6160, 6720,
7280, 9500, 9600, 9700, 9800 };
int dMap[18] = { 14, 28, 42, 56, 70, 84, 98, 112, 126, 140, 154, 168, 182, 196, 240, 250,
260, 270 };



byte MATenrich[8] = {106, 104, 102, 100, 98, 97, 95, 93};
int crankEnrich[8] =   {120, 120, 120, 110, 100, 100, 100, 100};
int afterStart[8] =  {135, 130, 120, 117, 115, 100, 100, 100};
int warmUp[8] =    {210, 160, 145, 135, 130, 126, 100, 100};

int deadTime[5] = { 1400, 1200, 1000, 800, 600 };

int  injstart[8] = {200, 200, 200, 200, 200, 200, 200, 200};
byte  fuelTrimMult[8] = {100, 100, 100, 100, 100, 100, 100, 100};
int  fuelTrimAdd[8] = {0, 0, 0, 0, 0, 0, 0, 0};

byte idleDC[8] = {0, 0, 0, 0, 0, 0, 0, 0};

byte boostTarget[8][8] = {

{2,  2,  3,  10, 13, 13, 10, 14 },
{2,  2,  3,  10, 13, 13, 10, 14 },
{2,  2,  3,  10, 13, 13, 10, 14 },
{2,  2,  3,  10, 13, 13, 10, 14 },
{2,  2,  3,  10, 13, 13, 10, 14 },
```

```
{2,  2,  3,  10, 13, 13, 10, 14 },
{2,  2,  3,  10, 13, 13, 10, 14 },
{2,  2,  3,  10, 13, 13, 10, 14 }

};


byte tpsdot[8] = {5, 10, 15, 20, 25, 25, 25, 25};
byte rpmtps[8] = {100, 100, 100, 100, 100, 100, 100, 100};
byte CLNTtps[8] = {100, 100, 100, 100, 100, 100, 100, 100};

int clntemp[10] = { 110, 85, 68, 57, 46, 36, 26, 15, 1, -30 };
int mattemp[10] = { 110, 85, 68, 57, 46, 36, 26, 15, 1, -30 };

const int cylAngle1=0, cylAngle2=540, cylAngle3=180, cylAngle4=360, cylAngle5=-5000,
cylAngle6=-5000;   // redosljed paljenja, cilindar kojeg nema se piše -5000
const byte injectorOrder[6] = {3, 1, 4, 2, 6, 6};    // stvarni redosljed ubrizg ( inj1,
inj3, inj4, inj2 )


////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////PINOVI//////////////////////////
//////////////////////////////////////////////////////////

const int
MAPPin=PA0,TPSpin=PC5,MATpin=PA2,CLNTpin=PA3,AFRpin=PC1,BATTpin=PA7,TPS2pin=PA1,nbAFRpin=PA4;
const int injPin1=PE0,injPin2=PB6,injPin3=PB4,injPin4=PD7,injPin5=PE2,injPin6=PE4;
int IGNpin1=PE3,IGNpin2=PE3,IGNpin3=PE3,IGNpin4=PE3,IGNpin5=PE3,IGNpin6=PE3,IGNpin;
const int
clutchPin=PE11,CKPpin=PB2,CAMpin=PE9,TBpin=PD15,NBheatpin=PA8,fuelPin=PD5,VTECpin=PC13,WIpin=P
E5,IACpin=PB7,EBCpin=PD3,BUTTONpin=PE13;

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////


int tps1CAL1,tps1CAL2,AFRcal,BATTcal;


byte Ytemp1,Ytemp2,limit=0,brprint=0;
byte crankEnrichVal,afterStartVal,warmUpVal,MATenrichVal;
int afterStartTime=1,crankRPM=400;

float reqFuel=0.7;
int fuelMap=1;
int Tinj1=1,Tinj2=1,Tinj3=1,Tinj4=1,Tinj5=1,Tinj6=1,VE,accPW,injDead,injector=0,tempTinj;
int TPSchange,TPStresh=2000,accPWstart,decay=200;
unsigned int PW,accPWtime,accPWsplit;
int rpmCoastLow=1500,rpmCoastHigh=4000,kPaCoast=18,coastStart,CoastCutDelay=1500;

int fi, Tfi, dwell=3000, fiCorr=-35, fiBoost=0;

int
chargeNZ1,chargeNZ2,chargeNZ3,chargeNZ4,chargeNZ5,chargeNZ6,fireNZ1,fireNZ2,fireNZ3,fireNZ4,fi
reNZ5,fireNZ6,fireNZ,chargeNZ;
int injAngle,inj1Angle,inj2Angle,inj3Angle,inj4Angle,inj5Angle,inj6Angle,
inj1Sch,inj2Sch,inj3Sch,inj4Sch,inj5Sch,inj6Sch, injSchMin=5,injSchMax=100;
int
Tcharge1,Tcharge2,Tcharge3,Tcharge4,Tcharge5,Tcharge6,Tfire1,Tfire2,Tfire3,Tfire4,Tfire5,Tfire
6;
int prevTcharge,prevTfire;
int
chargePos1,chargePos2,chargePos3,chargePos4,chargePos5,chargePos6,firePos1,firePos2,firePos3,f
irePos4,firePos5,firePos6;
int prevchargePos,prevfirePos;
int chargeIndex,fireIndex,nextCoilCharge=0,nextCoilFire=0;
volatile int coilChargeSch,coilFireSch;
```

```
int X1,X2,Y1,Y2,ZI,ZII,x,y,Z1,Z2,Z3,Z4;
float X,Y,Ytemp;
byte fX,fY,dX,dY;

int MAPsum,MAPsamples,MAPsamplePos;

int AFR,prevAFR,targetAFR,kPa,nbAFR,prevnbAFR;
int MAPVal,CLNTVal,MATVal,BATTVal,TPSVal,TPSValRaw,prevTPSVal=0,TPS2Val,TPSperc,TPS2perc;
int matval[5],clntval[5],battval[5],prevTPSValRaw;


unsigned int
RPM=15,avgRPM,LaunchHardCut=3500,LaunchSoftCut=3200,revLimit=7000,softRevLimit=7000,revLimitTi
ming=0,launchTiming=0,LaunchTPS=75;
unsigned int TPStime=0,lastprint=0,lastprint2,lastSerial=0,currentT,engineON,enginestart=0;



int GMT1=11,GMT2,GMT3,GMT4,GMT5,GMT6;
int
toothGap=30,teethNum=12,teethNumX2,missingTooth=2,teethNum_1,teethNumX2_1,teethNumX2_cyl,cylin
ders=4;
int crankAngle,prevcrankpos;
volatile int b1,b5=0,crankpos=150,i,ignEvent=0,Synced=0,CAMsync=0,nextInjOpen=20;
volatile int prevRPMsplit=100000000,triggersplit=0;
int ignEventEGO,ignEventMAP;

volatile unsigned int btrigger=0,RPMsplit=100000,RPMtime=0,myT=0,CAMfilter,ciclusTime;
unsigned int RPMsplit1,RPMtime1;


boolean ECUsend=0,ECUsendSlow=0,MapSync=0,EEPROMWrite=0,EEPROMRead=0;



int kpaWI=150,WItest=0;
int EBCdc;
int IACdc=0,IACdcHigh,IACdcLow,IACperiod=2000,IACstate=0,idleTarget=950;
int antiLag=0;

/////////PID///////

int Setpoint, Input, Ireset, device;
float iacKp=0.1, iacKi=0, ebcKp=1, ebcKi=0.007;


//////////////////////////TEST variables//////////////////////////
volatile unsigned int
inj1O=0,inj1C=0,injO=0,injC=0,cC=0,cF=0,GMTtime,GMTtime1,CAMsignal,error=0;
int IGNtest,motor=65500;
unsigned int sv,vrijeme1,vrijeme2;
////////////////////////////////////////////////////////////////////




HardwareTimer timerI(1);
HardwareTimer timerII(2);
HardwareTimer timerT(3);
HardwareTimer timerIV(4);
HardwareTimer timerV(5);



void setup() {

  Serial1.begin(1000000);
  Serial3.begin(250000);

  SPI_2.begin(); //Initialize the SPI_2 port.
  SPI_2.setBitOrder(MSBFIRST); // Set the SPI_2 bit order
```

```
  SPI_2.setDataMode(SPI_MODE0); //Set the  SPI_2 data mode 0
  SPI_2.setClockDivider(SPI_CLOCK_DIV32);  // Use a different speed to SPI 1


  pinModeSetup();


  teethNumX2=teethNum*2;
  teethNum_1=teethNum-1;
  teethNumX2_1=teethNumX2-1;
  teethNumX2_cyl=teethNumX2/cylinders;

  GMT2=GMT1+cylAngle2/toothGap; if ( GMT2>teethNumX2 ) { GMT2=GMT2-teethNumX2; }
  GMT3=GMT1+cylAngle3/toothGap; if ( GMT3>teethNumX2 ) { GMT3=GMT3-teethNumX2; }
  GMT4=GMT1+cylAngle4/toothGap; if ( GMT4>teethNumX2 ) { GMT4=GMT4-teethNumX2; }
  GMT5=GMT1+cylAngle5/toothGap; if ( GMT5>teethNumX2 ) { GMT5=GMT5-teethNumX2; }
  GMT6=GMT1+cylAngle6/toothGap; if ( GMT6>teethNumX2 ) { GMT6=GMT6-teethNumX2; }


  MAPsamplePos=GMT1;


  timerSETUP();  //setup all timers

  EEPROMRead=1;

}




void loop() {

 if ( GMTtime>0 ){ GMTtime1=GMTtime; GMTtime=0; Serial1.println(GMTtime1);  }
 //if ( GMTtime>0 ){ GMTtime1=GMTtime; GMTtime=0; Serial1.print(GMTtime1); Serial1.print("
 "); Serial1.println(CAMsignal);  }



////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                                  //Analog read
////////////////////////////////////////////////////////////////////////////////////////////
////////////

 TPSValRaw=analogRead(TPSpin);
 TPSVal=1.5842*TPSValRaw-741;
 if ( TPSVal<0 ){ TPSVal=0; }
 TPSperc=0.025*TPSVal;


 TPS2Val=analogRead(TPS2pin);
 TPS2perc=-0.2217*TPS2Val+749;


 AFR=analogRead(AFRpin);
 AFR=prevAFR*0.9+AFR*0.1;
 prevAFR=AFR;
 AFR=(AFR*0.00405+6.4)*10;    //7.35


 nbAFR=analogRead(nbAFRpin);
 nbAFR=prevnbAFR*0.9+nbAFR*0.1;
 prevnbAFR=nbAFR;
 nbAFR=nbAFR*0.8057;



 if ( ignEventMAP==1 ) {
  if ( MAPsamples==0 ) { MAPsamples=1; }
  MAPVal=MAPsum/MAPsamples;
```

```
 kPa= 0.0804*MAPVal-1;
 MAPsum=0; MAPsamples=0;
 ignEventMAP=0;
}
MAPVal=analogRead(MAPPin);
MAPsum=MAPsum+MAPVal;
MAPsamples=MAPsamples+1;
if ( RPM<400 ) {
 kPa= 0.0804*MAPVal-1;
 if ( Synced==0 ){ MAPsum=0; MAPsamples=0; }
 }



MATVal=analogRead(MATpin);
matval[4]=matval[3]; matval[3]=matval[2]; matval[2]=matval[1]; matval[1]=matval[0];
matval[0]=MATVal;
MATVal=( matval[4]+matval[3]+matval[2]+matval[1]+matval[0] )/5 ;
Ytemp=MATVal*0.0025-1;
if ( Ytemp<0.01 ){ Ytemp=0.01; }
if ( Ytemp>8.99 ){ Ytemp=8.99; }
Ytemp1=Ytemp; Ytemp2=Ytemp1+1;
MATVal=mattemp[Ytemp1]*(Ytemp2-Ytemp)+mattemp[Ytemp2]*(Ytemp-Ytemp1);    //Air temp

Ytemp=0.06667*MATVal+1.33;
if ( Ytemp<0.01 ){ Ytemp=0.01; }
if ( Ytemp>6.99 ){ Ytemp=6.99; }
Ytemp1=Ytemp; Ytemp2=Ytemp1+1;
MATenrichVal=MATenrich[Ytemp1]*(Ytemp2-Ytemp)+MATenrich[Ytemp2]*(Ytemp-Ytemp1);
/*MATcorr=-0.031*MATVal+21.26;
if ( MATcorr>13 ) { MATcorr=13; }
if ( MATcorr<0 )  { MATcorr=0; }*/



CLNTVal=analogRead(CLNTpin);
clntval[4]=clntval[3]; clntval[3]=clntval[2]; clntval[2]=clntval[1]; clntval[1]=clntval[0];
clntval[0]=CLNTVal;
CLNTVal=( clntval[4]+clntval[3]+clntval[2]+clntval[1]+clntval[0] )/5 ;
Ytemp=CLNTVal*0.0025-1;
if ( Ytemp<0.01 ){ Ytemp=0.01; }
if ( Ytemp>8.99 ){ Ytemp=8.99; }
Ytemp1=Ytemp; Ytemp2=Ytemp1+1;
CLNTVal=clntemp[Ytemp1]*(Ytemp2-Ytemp)+clntemp[Ytemp2]*(Ytemp-Ytemp1);     //Coolant temp

Ytemp=0.06667*CLNTVal+1.33;
if ( Ytemp<0.01 ){ Ytemp=0.01; }
if ( Ytemp>6.99 ){ Ytemp=6.99; }
Ytemp1=Ytemp; Ytemp2=Ytemp1+1;
if ( engineON < 10 ){
crankEnrichVal=crankEnrich[Ytemp1]*(Ytemp2-Ytemp)+crankEnrich[Ytemp2]*(Ytemp-Ytemp1);
afterStartVal=afterStart[Ytemp1]*(Ytemp2-Ytemp)+afterStart[Ytemp2]*(Ytemp-Ytemp1);
}
else { crankEnrichVal=100; afterStartVal=100; }

warmUpVal=warmUp[Ytemp1]*(Ytemp2-Ytemp)+warmUp[Ytemp2]*(Ytemp-Ytemp1);
if ( engineON>1000 && warmUpVal>107 ){ warmUpVal=107; }



BATTVal=analogRead(BATTpin);
BATTVal = BATTVal*0.544;
battval[4]=battval[3]; battval[3]=battval[2]; battval[2]=battval[1]; battval[1]=battval[0];
battval[0]=BATTVal;
BATTVal = ( battval[4]+battval[3]+battval[2]+battval[1]+battval[0] )/5 ;
if ( BATTVal<400 || BATTVal>1900 ) { BATTVal=1450; }
Ytemp=BATTVal*0.005-4;
if ( Ytemp<0.01 ) { Ytemp=0.01; }
if ( Ytemp>3.99 ) { Ytemp=3.99; }
Ytemp1=Ytemp;  Ytemp2=Ytemp1+1;
```

```
 injDead=deadTime[Ytemp1]*(Ytemp2-Ytemp)+deadTime[Ytemp2]*(Ytemp-Ytemp1);




////////////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                                //Serial to PC
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////

SerialPC();

EEPROMsync();


////////////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                                //Test
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////

/*
if ( inj1O==1 ){ Serial1.print("O1 "); Serial1.println(injO); inj1O=0; }
if ( inj1O==2 ){ Serial1.print("O2 "); Serial1.println(injO); inj1O=0; }
if ( inj1O==3 ){ Serial1.print("O3 "); Serial1.println(injO); inj1O=0; }
if ( inj1O==4 ){ Serial1.print("O4 "); Serial1.println(injO); inj1O=0; }
if ( inj1O==5 ){ Serial1.print("O5 "); Serial1.println(injO); inj1O=0; }
if ( inj1O==6 ){ Serial1.print("O6 "); Serial1.println(injO); inj1O=0; }

if ( inj1C==1 ){ Serial1.print("C1 "); Serial1.println(injC); inj1C=0; }
if ( inj1C==2 ){ Serial1.print("C2 "); Serial1.println(injC); inj1C=0; }
if ( inj1C==3 ){ Serial1.print("C3 "); Serial1.println(injC); inj1C=0; }
if ( inj1C==4 ){ Serial1.print("C4 "); Serial1.println(injC); inj1C=0; }
if ( inj1C==5 ){ Serial1.print("C5 "); Serial1.println(injC); inj1C=0; }
if ( inj1C==6 ){ Serial1.print("C6 "); Serial1.println(injC); inj1C=0; }
*/


if ( vrijeme2>0 ){  Serial1.print(vrijeme2); Serial1.println(" cam");  vrijeme2=0;}

if ( GMTtime>0 ){ GMTtime1=GMTtime; GMTtime=0; Serial1.println(GMTtime1); }
//if ( GMTtime>0 ){ GMTtime1=GMTtime; GMTtime=0; Serial1.print(GMTtime1); Serial1.print("
"); Serial1.println(CAMsignal);  }

if ( IGNtest==1 ) {
  IGNtest=0;
  digitalWrite(IGNpin1,HIGH); delay(3); digitalWrite(IGNpin1,LOW);
  delay(2000);
  digitalWrite(IGNpin2,HIGH); delay(3); digitalWrite(IGNpin2,LOW);
}



////////////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                          // RPM, revlimit, FFS...
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////

  noInterrupts(); RPMsplit1=RPMsplit; interrupts();
  noInterrupts(); RPMtime1=RPMtime; prevcrankpos=crankpos; interrupts();
  RPM=5000000/RPMsplit1;

  if ( ciclusTime>0 ){
    static unsigned int prevciclusTime,ciclusSplit;

    ciclusSplit=ciclusTime-prevciclusTime;
    prevciclusTime=ciclusTime;
    ciclusTime=0;
    avgRPM=120000000/ciclusSplit;
```

```
  }

  if ( RPM>350 && enginestart==0 ) { enginestart=millis(); }
  engineON=( millis()-enginestart )*0.001;

  noInterrupts(); currentT=myT+timerT.getCount(); interrupts();
  if ( (currentT-RPMtime1)>1000000 ) {
    RPM=15; enginestart=0; engineON=0; crankpos=150; btrigger=0; Synced=0;
    prevRPMsplit=100000000;
    nextCoilCharge==0; nextCoilFire==0;
    }

  if ( RPM>50 || millis()<2000 ) { digitalWrite(fuelPin,HIGH); }    //Fuel pump
    else { digitalWrite(fuelPin,LOW); }

  if ( digitalRead(clutchPin)==LOW && RPM>LaunchHardCut && TPSperc>LaunchTPS ){ limit=1;
  }      //Rev limit
    else{  if ( RPM>revLimit ) { limit=1; }
              else if ( limit==1 ) { limit=0; }
        }

  if ( digitalRead(BUTTONpin)==LOW ){ antiLag=1; }
    else{ antiLag=0; }


  if ( ignEvent==1 ){
    ignEvent=0;
    ignEventEGO=ignEventEGO+1;
    ignEventMAP=1;
  }


////////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                        // VTEC, WI, IAC
////////////////////////////////////////////////////////////////////////////////////////////////
////////////

VTECcalc();

WIcalc();

IACcalc();

EBCcalc();


////////////////////////////////////////////////////////////////////////////////////////////////
//////////
                                          //Injection PW
////////////////////////////////////////////////////////////////////////////////////////////////
//////////

  VEcalc();

  VE=VE*warmUpVal*0.01;
  VE=VE*MATenrichVal*0.01;
  if ( engineON < afterStartTime )  { VE=VE*afterStartVal*0.01; }
  if ( RPM < crankRPM )  { VE=VE*crankEnrichVal*0.01; }

  if ( RPM<rpmCoastLow || RPM>rpmCoastHigh || kPa>kPaCoast ){ coastStart=millis(); }
  if ( millis()-coastStart>CoastCutDelay ){ VE=0; }

  PW=reqFuel*VE*kPa+injDead;

  if ( RPM<400 && TPSperc>95 ){ PW=100; }          //Flood clear mode

  if ( PW<100 ){ PW=100; }
  //PW=14000; accPW=0;
```

```
  tempTinj=(PW*fuelTrimMult[0]*0.01) + accPW + fuelTrimAdd[0]*30;
  noInterrupts(); Tinj1=tempTinj; interrupts();

  tempTinj=(PW*fuelTrimMult[1]*0.01) + accPW + fuelTrimAdd[1]*30;
  noInterrupts(); Tinj2=tempTinj; interrupts();

  tempTinj=(PW*fuelTrimMult[2]*0.01) + accPW + fuelTrimAdd[2]*30;
  noInterrupts(); Tinj3=tempTinj; interrupts();

  tempTinj=(PW*fuelTrimMult[3]*0.01) + accPW + fuelTrimAdd[3]*30;
  noInterrupts(); Tinj4=tempTinj; interrupts();

  tempTinj=(PW*fuelTrimMult[4]*0.01) + accPW + fuelTrimAdd[4]*30;
  noInterrupts(); Tinj5=tempTinj; interrupts();

  tempTinj=(PW*fuelTrimMult[5]*0.01) + accPW + fuelTrimAdd[5]*30;
  noInterrupts(); Tinj6=tempTinj; interrupts();



////////////////////////////////////////////////////////////////////////////////////////////////////
//////////
                                          //Injection angle
////////////////////////////////////////////////////////////////////////////////////////////////////
//////////

  Ytemp=0.000667*RPM-0.66667;
  if ( Ytemp<0.01 ){ Ytemp=0.01; }
  if ( Ytemp>6.99 ){ Ytemp=6.99; }
  Ytemp1=Ytemp; Ytemp2=Ytemp1+1;
  injAngle=injstart[Ytemp1]*(Ytemp2-Ytemp)+injstart[Ytemp2]*(Ytemp-Ytemp1);
  if ( injAngle < 0 ){ injAngle=0; }
  if ( injAngle > 719 ){ injAngle=719; }


  inj1Angle=cylAngle1+injAngle; if ( inj1Angle>719 ){ inj1Angle=inj1Angle-720; }
  inj2Angle=cylAngle2+injAngle; if ( inj2Angle>719 ){ inj2Angle=inj2Angle-720; }
  inj3Angle=cylAngle3+injAngle; if ( inj3Angle>719 ){ inj3Angle=inj3Angle-720; }
  inj4Angle=cylAngle4+injAngle; if ( inj4Angle>719 ){ inj4Angle=inj4Angle-720; }
  inj5Angle=cylAngle5+injAngle; if ( inj5Angle>719 ){ inj5Angle=inj5Angle-720; }
  inj6Angle=cylAngle6+injAngle; if ( inj6Angle>719 ){ inj6Angle=inj6Angle-720; }

  noInterrupts(); currentT=myT+timerT.getCount(); interrupts();

if ( prevcrankpos<teethNumX2 ){

  if ( prevcrankpos<GMT1 ) { crankAngle=(teethNumX2+prevcrankpos-GMT1)*toothGap; }
    else { crankAngle=(prevcrankpos-GMT1)*toothGap; }
  crankAngle=crankAngle+((currentT-RPMtime1)*toothGap)/RPMsplit1;


  if ( crankAngle>inj1Angle ) { inj1Sch=inj1Angle+720-crankAngle; }
  else { inj1Sch=inj1Angle-crankAngle; }
  if ( inj1Sch>injSchMin && inj1Sch<injSchMax && (nextInjOpen==1 || nextInjOpen==20) ){
      inj1Sch=(inj1Sch*RPMsplit1)/toothGap;
      if ( inj1Sch<65000 ){
        uint16_t timerCCR;
        timerCCR = TIMER2->regs.gen->CNT + inj1Sch;
        noInterrupts();
        TIMER2->regs.gen->CCR3 = timerCCR;
        injector=1;
        nextInjOpen==1;
        interrupts();
      }
     }

  if ( crankAngle>inj2Angle ) { inj2Sch=inj2Angle+720-crankAngle; }
  else { inj2Sch=inj2Angle-crankAngle; }
  if ( inj2Sch>injSchMin && inj2Sch<injSchMax && (nextInjOpen==2 || nextInjOpen==20) ){
      inj2Sch=(inj2Sch*RPMsplit1)/toothGap;
      if ( inj2Sch<65000 ){
```

```
        uint16_t timerCCR;
        timerCCR = TIMER2->regs.gen->CNT + inj2Sch;
        noInterrupts();
        TIMER2->regs.gen->CCR3 = timerCCR;
        injector=2;
        nextInjOpen=2;
        interrupts();
      }
     }

  if ( crankAngle>inj3Angle ) { inj3Sch=inj3Angle+720-crankAngle; }
  else { inj3Sch=inj3Angle-crankAngle; }
  if ( inj3Sch>injSchMin && inj3Sch<injSchMax && (nextInjOpen==3 || nextInjOpen==20) ){
     inj3Sch=(inj3Sch*RPMsplit1)/toothGap;
     if ( inj3Sch<65000 ){
        uint16_t timerCCR;
        timerCCR = TIMER2->regs.gen->CNT + inj3Sch;
        noInterrupts();
        TIMER2->regs.gen->CCR3 = timerCCR;
        injector=3;
        nextInjOpen=3;
        interrupts();
      }
     }

  if ( crankAngle>inj4Angle ) { inj4Sch=inj4Angle+720-crankAngle; }
  else { inj4Sch=inj4Angle-crankAngle; }
  if ( inj4Sch>injSchMin && inj4Sch<injSchMax && (nextInjOpen==4 || nextInjOpen==20) ){
     inj4Sch=(inj4Sch*RPMsplit1)/toothGap;
     if ( inj4Sch<65000 ){
        uint16_t timerCCR;
        timerCCR = TIMER2->regs.gen->CNT + inj4Sch;
        noInterrupts();
        TIMER2->regs.gen->CCR3 = timerCCR;
        injector=4;
        nextInjOpen=4;
        interrupts();
      }
     }

  if ( crankAngle>inj5Angle ) { inj5Sch=inj5Angle+720-crankAngle; }
  else { inj5Sch=inj5Angle-crankAngle; }
  if ( inj5Sch>injSchMin && inj5Sch<injSchMax && (nextInjOpen==5 || nextInjOpen==20) ){
     inj5Sch=(inj5Sch*RPMsplit1)/toothGap;
     if ( inj5Sch<65000 ){
        uint16_t timerCCR;
        timerCCR = TIMER2->regs.gen->CNT + inj5Sch;
        noInterrupts();
        TIMER2->regs.gen->CCR3 = timerCCR;
        injector=5;
        nextInjOpen=5;
        interrupts();
      }
     }

  if ( crankAngle>inj6Angle ) { inj6Sch=inj6Angle+720-crankAngle; }
  else { inj6Sch=inj6Angle-crankAngle; }
  if ( inj6Sch>injSchMin && inj6Sch<injSchMax && (nextInjOpen==6 || nextInjOpen==20) ){
     inj6Sch=(inj6Sch*RPMsplit1)/toothGap;
     if ( inj6Sch<65000 ){
        uint16_t timerCCR;
        timerCCR = TIMER2->regs.gen->CNT + inj6Sch;
        noInterrupts();
        TIMER2->regs.gen->CCR3 = timerCCR;
        injector=6;
        nextInjOpen=6;
        interrupts();
      }
     }

injSchMin=5;
```

```
injSchMax=360;

}



if ( GMTtime>0 ){ GMTtime1=GMTtime; GMTtime=0; Serial1.println(GMTtime1); }
//if ( GMTtime>0 ){ GMTtime1=GMTtime; GMTtime=0; Serial1.print(GMTtime1); Serial1.print("
"); Serial1.println(CAMsignal);  }

////////////////////////////////////////////////////////////////////////////////////////////
//////////
                                          //Ignition timing
////////////////////////////////////////////////////////////////////////////////////////////
//////////

   IgnCalc();

   dwell=3000;
   if ( RPM > softRevLimit ) { fi=revLimitTiming; }
   if ( digitalRead(clutchPin)==LOW && RPM>LaunchSoftCut && TPSperc>95 ){ fi=launchTiming; }
   if ( RPM < 350 && engineON < 8 ) { dwell=4500; }
   if ( CLNTVal<50 && kPa<30 ) {
    fi=fi+3;
    if ( CLNTVal<35 ) { fi=fi+3; }
   }
   //fi=6; fiCorr=0;
   if ( antiLag==1 && TPSperc<5 ){ fi=-30; }

   Tfi=RPMsplit1*0.0333*(fi+fiCorr);

////////////////////////////////////////Coil charging
calc////////////////////////////////////////

   chargeNZ=(fi+fiCorr)/toothGap+dwell/RPMsplit1+2;

   chargePos1=GMT1-chargeNZ; chargeNZ1=chargeNZ;
   if ( chargePos1<1 ) { chargePos1=teethNumX2+chargePos1; }

   chargePos2=GMT2-chargeNZ; chargeNZ2=chargeNZ;
   if ( chargePos2<1 ) { chargePos2=teethNumX2+chargePos2; }

   chargePos3=GMT3-chargeNZ; chargeNZ3=chargeNZ;
   if ( chargePos3<1 ) { chargePos3=teethNumX2+chargePos3; }

   chargePos4=GMT4-chargeNZ; chargeNZ4=chargeNZ;
   if ( chargePos4<1 ) { chargePos4=teethNumX2+chargePos4; }

   chargePos5=GMT5-chargeNZ; chargeNZ5=chargeNZ;
   if ( chargePos5<1 ) { chargePos5=teethNumX2+chargePos5; }

   chargePos6=GMT6-chargeNZ; chargeNZ6=chargeNZ;
   if ( chargePos6<1 ) { chargePos6=teethNumX2+chargePos6; }


   Tcharge1=RPMsplit1*chargeNZ1-Tfi-dwell;
   if ( Tcharge1>65000 ){ Tcharge1=65000; }
   if ( Tcharge1<3 ) { Tcharge1=3; }

   Tcharge2=RPMsplit1*chargeNZ2-Tfi-dwell;
   if ( Tcharge2>65000 ){ Tcharge2=65000; }
   if ( Tcharge2<3 ) { Tcharge2=3; }

   Tcharge3=RPMsplit1*chargeNZ3-Tfi-dwell;
   if ( Tcharge3>65000 ){ Tcharge3=65000; }
   if ( Tcharge3<3 ) { Tcharge3=3; }

   Tcharge4=RPMsplit1*chargeNZ4-Tfi-dwell;
   if ( Tcharge4>65000 ){ Tcharge4=65000; }
   if ( Tcharge4<3 ) { Tcharge4=3; }
```

```
   Tcharge5=RPMsplit1*chargeNZ5-Tfi-dwell;
   if ( Tcharge5>65000 ){ Tcharge5=65000; }
   if ( Tcharge5<3 ) { Tcharge5=3; }

   Tcharge6=RPMsplit1*chargeNZ6-Tfi-dwell;
   if ( Tcharge6>65000 ){ Tcharge6=65000; }
   if ( Tcharge6<3 ) { Tcharge6=3; }



//////////////////////////////////////////Coil fire
calc//////////////////////////////////////

   fireNZ=(fi+fiCorr)/toothGap+1;

   firePos1=GMT1-fireNZ; fireNZ1=fireNZ;
   if ( firePos1<1 ){ firePos1=teethNumX2+firePos1; }

   firePos2=GMT2-fireNZ; fireNZ2=fireNZ;
   if ( firePos2<1 ){ firePos2=teethNumX2+firePos2; }

   firePos3=GMT3-fireNZ; fireNZ3=fireNZ;
   if ( firePos3<1 ){ firePos3=teethNumX2+firePos3; }

   firePos4=GMT4-fireNZ; fireNZ4=fireNZ;
   if ( firePos4<1 ){ firePos4=teethNumX2+firePos4; }

   firePos5=GMT5-fireNZ; fireNZ5=fireNZ;
   if ( firePos5<1 ){ firePos5=teethNumX2+firePos5; }

   firePos6=GMT6-fireNZ; fireNZ6=fireNZ;
   if ( firePos6<1 ){ firePos6=teethNumX2+firePos6; }


   Tfire1=RPMsplit1*fireNZ1-Tfi;
   if ( Tfire1>65000 ){ Tfire1=65000; }
   if ( Tfire1<3 ) { Tfire1=3; }

   Tfire2=RPMsplit1*fireNZ2-Tfi;
   if ( Tfire2>65000 ){ Tfire2=65000; }
   if ( Tfire2<3 ) { Tfire2=3; }

   Tfire3=RPMsplit1*fireNZ3-Tfi;
   if ( Tfire3>65000 ){ Tfire3=65000; }
   if ( Tfire3<3 ) { Tfire3=3; }

   Tfire4=RPMsplit1*fireNZ4-Tfi;
   if ( Tfire4>65000 ){ Tfire4=65000; }
   if ( Tfire4<3 ) { Tfire4=3; }

   Tfire5=RPMsplit1*fireNZ5-Tfi;
   if ( Tfire5>65000 ){ Tfire5=65000; }
   if ( Tfire5<3 ) { Tfire5=3; }

   Tfire6=RPMsplit1*fireNZ6-Tfi;
   if ( Tfire6>65000 ){ Tfire6=65000; }
   if ( Tfire6<3 ) { Tfire6=3; }



   if ( nextCoilCharge==0 && Synced==1 ){

      if ( chargePos1>(crankpos+1) && chargePos1<(teethNumX2_cyl+crankpos) ){
      nextCoilCharge=1; nextCoilFire=1; }

      if ( chargePos2>(crankpos+1) && chargePos2<(teethNumX2_cyl+crankpos) ){
      nextCoilCharge=2; nextCoilFire=2; }

      if ( chargePos3>(crankpos+1) && chargePos3<(teethNumX2_cyl+crankpos) ){
      nextCoilCharge=3; nextCoilFire=3; }
```

```
   if ( chargePos4>(crankpos+1) && chargePos4<(teethNumX2_cyl+crankpos) ){
   nextCoilCharge=4; nextCoilFire=4; }

   if ( chargePos5>(crankpos+1) && chargePos5<(teethNumX2_cyl+crankpos) ){
   nextCoilCharge=5; nextCoilFire=5; }

   if ( chargePos6>(crankpos+1) && chargePos6<(teethNumX2_cyl+crankpos) ){
   nextCoilCharge=6; nextCoilFire=6; }

}


noInterrupts();
if ( nextCoilCharge==1 && ( chargePos1>crankpos || crankpos>(chargePos1+teethNum) ) ){
prevTcharge=Tcharge1; prevchargePos=chargePos1; IGNpin=IGNpin1; chargeIndex=0; }
interrupts();

noInterrupts();
if ( nextCoilCharge==2 && ( chargePos2>crankpos || crankpos>(chargePos2+teethNum) ) ){
prevTcharge=Tcharge2; prevchargePos=chargePos2; IGNpin=IGNpin2; chargeIndex=1; }
interrupts();

noInterrupts();
if ( nextCoilCharge==3 && ( chargePos3>crankpos || crankpos>(chargePos3+teethNum) ) ){
prevTcharge=Tcharge3; prevchargePos=chargePos3; IGNpin=IGNpin3; chargeIndex=2; }
interrupts();

noInterrupts();
if ( nextCoilCharge==4 && ( chargePos4>crankpos || crankpos>(chargePos4+teethNum) ) ){
prevTcharge=Tcharge4; prevchargePos=chargePos4; IGNpin=IGNpin4; chargeIndex=3; }
interrupts();

noInterrupts();
if ( nextCoilCharge==5 && ( chargePos5>crankpos || crankpos>(chargePos5+teethNum) ) ){
prevTcharge=Tcharge5; prevchargePos=chargePos5; IGNpin=IGNpin5; chargeIndex=4; }
interrupts();

noInterrupts();
if ( nextCoilCharge==6 && ( chargePos6>crankpos || crankpos>(chargePos6+teethNum) ) ){
prevTcharge=Tcharge6; prevchargePos=chargePos6; IGNpin=IGNpin6; chargeIndex=5; }
interrupts();



noInterrupts();
if ( nextCoilFire==1 && ( firePos1>crankpos || crankpos>(firePos1+teethNum) ) ){
prevTfire=Tfire1; prevfirePos=firePos1; IGNpin=IGNpin1; fireIndex=0; }
interrupts();

noInterrupts();
if ( nextCoilFire==2 && ( firePos2>crankpos || crankpos>(firePos2+teethNum) ) ){
prevTfire=Tfire2; prevfirePos=firePos2; IGNpin=IGNpin2; fireIndex=1; }
interrupts();

noInterrupts();
if ( nextCoilFire==3 && ( firePos3>crankpos || crankpos>(firePos3+teethNum) ) ){
prevTfire=Tfire3; prevfirePos=firePos3; IGNpin=IGNpin3; fireIndex=2; }
interrupts();

noInterrupts();
if ( nextCoilFire==4 && ( firePos4>crankpos || crankpos>(firePos4+teethNum) ) ){
prevTfire=Tfire4; prevfirePos=firePos4; IGNpin=IGNpin4; fireIndex=3; }
interrupts();

noInterrupts();
if ( nextCoilFire==5 && ( firePos5>crankpos || crankpos>(firePos5+teethNum) ) ){
prevTfire=Tfire5; prevfirePos=firePos5; IGNpin=IGNpin5; fireIndex=4; }
interrupts();

noInterrupts();
```

```
   if ( nextCoilFire==6 && ( firePos6>crankpos || crankpos>(firePos6+teethNum) ) ){
   prevTfire=Tfire6; prevfirePos=firePos6; IGNpin=IGNpin6; fireIndex=5; }
   interrupts();




////////////////////////////////////////////////////////////////////////////////////////////////////
//////////
                                        //Acceleration enrichment
////////////////////////////////////////////////////////////////////////////////////////////////////
//////////

  if ( micros()-TPStime > 15000 ) {
    static int tpsAllowed,TPSfact,rpmTPSfact,clntTPSfact;

    TPSchange=(TPSVal-prevTPSVal)*67;
    if ( TPSchange > TPStresh ) { tpsAllowed=tpsAllowed+1; }
    if ( TPSchange < TPStresh ) { tpsAllowed=0; }

    if ( TPSchange > TPStresh && tpsAllowed==3 ) {
      Ytemp=TPSchange*0.0005;
      if ( Ytemp<0.01 ){ Ytemp=0.01; }
      if ( Ytemp>6.99 ){ Ytemp=6.99; }
      Ytemp1=Ytemp; Ytemp2=Ytemp1+1;
      TPSfact= tpsdot[Ytemp1]*(Ytemp2-Ytemp)+tpsdot[Ytemp2]*(Ytemp-Ytemp1);

      Ytemp=0.001*RPM-1;
      if ( Ytemp<0.01 ){ Ytemp=0.01; }
      if ( Ytemp>6.99 ){ Ytemp=6.99; }
      Ytemp1=Ytemp; Ytemp2=Ytemp1+1;
      rpmTPSfact= rpmtps[Ytemp1]*(Ytemp2-Ytemp)+rpmtps[Ytemp2]*(Ytemp-Ytemp1);

      Ytemp=0.06667*CLNTVal+1.33;
      if ( Ytemp<0.01 ){ Ytemp=0.01; }
      if ( Ytemp>6.99 ){ Ytemp=6.99; }
      Ytemp1=Ytemp; Ytemp2=Ytemp1+1;
      clntTPSfact= CLNTtps[Ytemp1]*(Ytemp2-Ytemp)+CLNTtps[Ytemp2]*(Ytemp-Ytemp1);

      accPWstart=TPSfact*rpmTPSfact*(clntTPSfact*0.01);
      accPWtime=millis();
      tpsAllowed=0;
    }

    if ( tpsAllowed==0 ){ prevTPSVal=TPSVal; TPStime=micros(); }

  }

  accPWsplit=millis()-accPWtime;
  if ( accPWsplit>2000 ) { accPWsplit=2000; }
  accPW=(accPWstart*accPWsplit)/decay;
  accPW=-accPW+accPWstart;
  if ( accPW < 0 ) { accPW=0; }
  if ( accPW > 20000 ) { accPW=20000; }

  //if ( accPW>0 ){Serial1.print("tps="); Serial1.println(accPW);}



////////////////////////////////////////////////////////////////////////////////////////////////////
//////////
                                                  //EGO
////////////////////////////////////////////////////////////////////////////////////////////////////
//////////

targetAFR=147;

if ( kPa<90 && RPM>100000 && CLNTVal>79 && AFR<200 && AFR>90 && accPW==0 ) {
  if ( ( RPM-(X1-1)*560 )<200 || ( (X2-1)*560-RPM )<200 ) {
      if ( (kPa-Y1*14)<5 || (Y2*14-kPa)<5 ) {
```

```
            if ( ignEventEGO>70 && AFR<targetAFR ) {
                ignEventEGO=0;
                f1[14-y][x-1]=f1[14-y][x-1]-1;
                if ( kPa>80 ) { f1[14-y][x-1]=f1[14-y][x-1]-1; }
               }
            if ( ignEventEGO>70 && AFR>(targetAFR+5) ) {
                ignEventEGO=0;
                f1[14-y][x-1]=f1[14-y][x-1]+1;
                if ( kPa>80 ) { f1[14-y][x-1]=f1[14-y][x-1]+1; }
               }
      }
      else { ignEventEGO=0; }
  }
  else { ignEventEGO=0; }
}
else{ ignEventEGO=0; }









////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////

}    //End of main loop







////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////



void pinModeSetup(){

  pinMode(injPin1, OUTPUT);
  pinMode(injPin2, OUTPUT);
  pinMode(injPin3, OUTPUT);
  pinMode(injPin4, OUTPUT);
  //pinMode(injPin5, OUTPUT);
  //pinMode(injPin6, OUTPUT);

  pinMode(IGNpin1, OUTPUT);
  /*pinMode(IGNpin2, OUTPUT);
  pinMode(IGNpin3, OUTPUT);
  pinMode(IGNpin4, OUTPUT);
  pinMode(IGNpin5, OUTPUT);
  pinMode(IGNpin6, OUTPUT);*/


  pinMode(clutchPin,INPUT_PULLUP);
  pinMode(BUTTONpin,INPUT_PULLUP);
  pinMode(CKPpin, INPUT);
```

```
  pinMode(CAMpin, INPUT);
  pinMode(fuelPin, OUTPUT);
  pinMode(VTECpin, OUTPUT); digitalWrite(VTECpin,HIGH);
  pinMode(WIpin, OUTPUT);
  pinMode(IACpin, OUTPUT);
  pinMode(EBCpin, OUTPUT);


  //pinMode(TBpin, OUTPUT); digitalWrite(TBpin,HIGH);
  /*pinMode(NBheatpin, PWM); pinMode(NBheatpin, OUTPUT); digitalWrite(NBheatpin, HIGH);*/
  pinMode(SPI2_NSS_PIN, OUTPUT); digitalWrite(SPI2_NSS_PIN,HIGH);

  pinMode(TPSpin, INPUT_ANALOG);
  pinMode(AFRpin, INPUT_ANALOG);
  pinMode(MAPPin, INPUT_ANALOG);
  pinMode(MATpin, INPUT_ANALOG);
  pinMode(BATTpin, INPUT_ANALOG);
  pinMode(CLNTpin, INPUT_ANALOG);


  delay(10);

  attachInterrupt(CKPpin,IGN,FALLING);
  attachInterrupt(CAMpin,CAMtrigg,RISING);

}



////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////



void timerSETUP(){

noInterrupts();

//set timerIV
  timerIV.pause();
  timerIV.setPrescaleFactor(84);
  timerIV.setOverflow(65535);
  timerIV.setMode(1, TIMER_OUTPUT_COMPARE);  //inj1 close
  timerIV.setMode(2, TIMER_OUTPUT_COMPARE);  //inj2 close
  timerIV.setMode(3, TIMER_OUTPUT_COMPARE);  //inj3 close
  timerIV.setMode(4, TIMER_OUTPUT_COMPARE);  //inj4 close
  timerIV.refresh();
  timerIV.resume();

//set timerII
  timerII.pause();
  timerII.setPrescaleFactor(84);
  timerII.setOverflow(65535);
  timerII.setMode(1, TIMER_OUTPUT_COMPARE);   //inj5 close
  timerII.setMode(2, TIMER_OUTPUT_COMPARE);   //inj6 close
  timerII.setMode(3, TIMER_OUTPUT_COMPARE);   //Injector open
  timerII.setMode(4, TIMER_OUTPUT_COMPARE);   //virtual trigger
  timerII.attachInterrupt(3,injOpen);
  timerII.refresh();
  timerII.resume();

//set timerI
  timerI.pause();
  timerI.setPrescaleFactor(168);
  timerI.setOverflow(65535);
```

```
  timerI.setMode(1, TIMER_OUTPUT_COMPARE);     //coil charge
  timerI.setMode(2, TIMER_OUTPUT_COMPARE);     //coil fire
  timerI.refresh();
  timerI.resume();


//set timerIII
  timerT.pause();
  timerT.setPrescaleFactor(84);
  timerT.setOverflow(65535);
  timerT.setMode(1, TIMER_OUTPUT_COMPARE);    //myTimer
  timerT.setCompare( 1, 60000 );
  timerT.attachInterrupt(1, myTimer);
  timerT.refresh();
  TIMER3->regs.gen->SR &= ~0x0002;
  timerT.resume();

//set timerV
  timerV.pause();
  timerV.setPrescaleFactor(84);
  timerV.setOverflow(65535);
  timerV.setMode(1, TIMER_OUTPUT_COMPARE);    //PWM
  timerV.refresh();
  timerV.resume();



  NVIC_BASE->IP[8] = 16;  //priority exti line 2   16
  NVIC_BASE->IP[27] = 0;  //priority timer 1 compare

interrupts();

}




void myTimer(void) {

   noInterrupts();
   myT=myT+TIMER3->regs.gen->CNT;
   TIMER3->regs.gen->CNT = 0;
   interrupts();

}



////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////



void SerialPC(){

/*if( micros()-lastprint>50000 ){

  Serial.print("mapRaw="); Serial.print(analogRead(MAPPin));
  Serial.print("tpsRaw="); Serial.print(analogRead(TPSpin));
  Serial.print("battRaw="); Serial.println(analogRead(BATTpin));

  lastprint=micros();

}
```

```
*/
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                                    //Receive data
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////
unsigned int serialReadTime;

serialReadTime=micros();

 while ( Serial1.available() > 1 ){

   static int val,row,column;
   static byte packetBegin,address,checkData;

   val=Serial1.peek();

   if ( val==255 ) {
    packetBegin=1;
    Serial1.read();
    address=Serial1.read();
   }
   else if ( packetBegin==0 ){
    Serial1.read();
   }


   if ( address==1 ){
    if ( Serial1.available() > 3 ){
      row=Serial1.read();
      column=Serial1.read();
      checkData=Serial1.read();
      if ( Serial1.read()==254 ){ f1[row][column]=checkData; }
      packetBegin=0; address=0;
    }
   }

   else if ( address==2 ){
    if ( Serial1.available() > 3 ){
      row=Serial1.read();
      column=Serial1.read();
      checkData=Serial1.read();
      if ( Serial1.read()==254 ){ f2[row][column]=checkData; }
      packetBegin=0; address=0;
    }
   }

   else if ( address==3 ){
    if ( Serial1.available() > 3 ){
      row=Serial1.read();
      column=Serial1.read();
      checkData=Serial1.read()-50;
      if ( Serial1.read()==254 ){ d[row][column]=checkData; }
      packetBegin=0; address=0;
    }
   }

   else if ( address==15 ){
    if ( Serial1.available() > 2 ){
      column=Serial1.read();
      crankEnrich[column]=Serial1.read()*2;
      Serial1.read();
      packetBegin=0; address=0;
    }
   }

   else if ( address==16 ){
    if ( Serial1.available() > 2 ){
      column=Serial1.read();
      afterStart[column]=Serial1.read()*2;
      Serial1.read();
```

```
    packetBegin=0; address=0;
   }
  }

  else if ( address==17 ){
   if ( Serial1.available() > 2 ){
    column=Serial1.read();
    warmUp[column]=Serial1.read()*2;
    Serial1.read();
    packetBegin=0; address=0;
   }
  }

  else if ( address==18 ){
   if ( Serial1.available() > 2 ){
    column=Serial1.read();
    MATenrich[column]=Serial1.read();
    Serial1.read();
    packetBegin=0; address=0;
   }
  }

  else if ( address==19 ){
   if ( Serial1.available() > 2 ){
    column=Serial1.read();
    tpsdot[column]=Serial1.read();
    Serial1.read();
    packetBegin=0; address=0;
   }
  }

  else if ( address==20 ){
   if ( Serial1.available() > 2 ){
    column=Serial1.read();
    rpmtps[column]=Serial1.read();
    Serial1.read();
    packetBegin=0; address=0;
   }
  }

  else if ( address==21 ){
   if ( Serial1.available() > 2 ){
    column=Serial1.read();
    CLNTtps[column]=Serial1.read()*2;
    Serial1.read();
    packetBegin=0; address=0;
   }
  }

  else if ( address==22 ){
   if ( Serial1.available() > 2 ){
    column=Serial1.read();
    fuelTrimMult[column]=Serial1.read();
    Serial1.read();
    packetBegin=0; address=0;
   }
  }

  else if ( address==23 ){
   if ( Serial1.available() > 2 ){
    column=Serial1.read();
    fuelTrimAdd[column]=Serial1.read()-100;
    Serial1.read();
    packetBegin=0; address=0;
   }
  }

  else if ( address==24 ){
   if ( Serial1.available() > 2 ){
    column=Serial1.read();
    injstart[column]=Serial1.read()*3;
```

```
    Serial1.read();
    packetBegin=0; address=0;
  }
 }

 else if ( address==10 ){
  if ( Serial1.available() > 1 ){
    crankRPM=Serial1.read()*3;
    Serial1.read();
    packetBegin=0; address=0;
  }
 }

 else if ( address==11 ){
  if ( Serial1.available() > 1 ){
    afterStartTime=Serial1.read();
    Serial1.read();
    packetBegin=0; address=0;
  }
 }

 else if ( address==12 ){
  if ( Serial1.available() > 2 ){
    decay=Serial1.read()*3;
    TPStresh=Serial1.read()*40;
    Serial1.read();
    packetBegin=0; address=0;
  }
 }

 else if ( address==13 ){
  if ( Serial1.available() > 4 ){
    rpmCoastLow=Serial1.read()*40;
    rpmCoastHigh=Serial1.read()*40;
    kPaCoast=Serial1.read();
    CoastCutDelay=Serial1.read()*100;
    Serial1.read();
    packetBegin=0; address=0;
  }
 }

 else if ( address==5 ){
  if ( Serial1.available() > 1 ){
    ECUsend=Serial1.read();
    Serial1.read();
    packetBegin=0; address=0;
  }
 }

 else if ( address==6 ){
  if ( Serial1.available() > 1 ){
    ECUsendSlow=Serial1.read();
    Serial1.read();
    packetBegin=0; address=0;
  }
 }

 else if ( address==14 ){
  if ( Serial1.available() > 3 ){
    revLimit=Serial1.read()*50;
    softRevLimit=Serial1.read()*50;
    revLimitTiming=Serial1.read()-50;
    Serial1.read();
    packetBegin=0; address=0;
  }
 }

 else if ( address==25 ){
  if ( Serial1.available() > 4 ){
    LaunchHardCut=Serial1.read()*50;
    LaunchSoftCut=Serial1.read()*50;
```

```
      launchTiming=Serial1.read()-50;
      LaunchTPS=Serial1.read();
      Serial1.read();
      packetBegin=0; address=0;
    }
   }

   else if ( address==26 ){
    if ( Serial1.available() > 2 ){
      column=Serial1.read();
      idleDC[column]=Serial1.read();
      Serial1.read();
      packetBegin=0; address=0;
    }
   }

   else if ( address==27 ){
    if ( Serial1.available() > 1 ){
      fiCorr=Serial1.read()-50;
      Serial1.read();
      packetBegin=0; address=0;
    }
   }

   else if ( address==8 ){
    if ( Serial1.available() > 1 ){
      EEPROMWrite=Serial1.read();
      Serial1.read();
      packetBegin=0; address=0;
    }
   }

   else if ( address==9 ){
    if ( Serial1.available() > 1 ){
      EEPROMRead=Serial1.read();
      Serial1.read();
      packetBegin=0; address=0;
    }
   }

   else if ( address==253 ){
    if ( Serial1.available() > 1 ){
      MapSync=Serial1.read();
      Serial1.read();
      packetBegin=0; address=0;
    }
   }

   else { packetBegin=0; address=0; }


if ( micros()-serialReadTime>500 ){ break; }
if ( Serial1.available()<6 ){ break; }

}




////////////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                                //Transmit data
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////

   if( micros()-lastprint>50000 && ECUsend==1 ){

       int tempSend;

       Serial1.write(255); Serial1.write(5);
```

```
      tempSend=RPM/250; Serial1.write(tempSend); Serial1.write(RPM-tempSend*250);
      Serial1.write(kPa);
      Serial1.write(AFR);
      Serial1.write(TPSperc+50);
      tempSend=PW/250; Serial1.write(tempSend); Serial1.write(PW-tempSend*250);
      Serial1.write(fi+50);
      Serial1.write(CLNTVal+50);
      Serial1.write(MATVal+50);
      Serial1.write(BATTVal/10);
      tempSend=accPW/250; Serial1.write(tempSend); Serial1.write(accPW-tempSend*250);
      Serial1.write(VE);
      Serial1.write(injDead/10);
      Serial1.write(injAngle/3);
      Serial1.write(1);
      Serial1.write(fX); Serial1.write(fY); Serial1.write(dX); Serial1.write(dY);
      Serial1.write(fuelMap);

      Serial1.write(254);
      lastprint=micros();

  }


  if( micros()-lastprint2>200000 && ECUsendSlow==1 ){
     int tempSend;

      Serial1.write(255); Serial1.write(6);

      Serial1.write(crankEnrichVal/2);
      Serial1.write(afterStartVal/2);
      Serial1.write(warmUpVal/2);
      Serial1.write(MATenrichVal);
      Serial1.write(IACdc);
      Serial1.write(EBCdc);

      Serial1.write(255);
      lastprint2=micros();
  }







////////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                                //ECUTune Sync
////////////////////////////////////////////////////////////////////////////////////////////////
////////////

    if ( MapSync==1 ) {
      Serial1.write(255); Serial1.write(1);
      for(int i=0; i<18; i++){
       for(int j=0; j<18; j++){
         Serial1.write(f1[i][j]);
        }
      }
      Serial1.write(254);

      Serial1.write(255); Serial1.write(2);
      for(int i=0; i<18; i++){
       for(int j=0; j<18; j++){
         Serial1.write(f2[i][j]);
        }
      }
      Serial1.write(254);

      Serial1.write(255); Serial1.write(3);
      for(int i=0; i<18; i++){
       for(int j=0; j<18; j++){
```

```
        Serial1.write(d[i][j]+50);
      }
    }
    Serial1.write(254);



    Serial1.write(255); Serial1.write(7);
    for(int i=0; i<18; i++){
        Serial1.write(dRpm[i]/50);
    }
    for(int i=0; i<18; i++){
        Serial1.write(dMap[i]/2);
    }
    for(int i=0; i<18; i++){
        Serial1.write(f1Rpm[i]/50);
    }
    for(int i=0; i<18; i++){
        Serial1.write(f1Map[i]/2);
    }
    for(int i=0; i<18; i++){
        Serial1.write(f2Rpm[i]/50);
    }
    for(int i=0; i<18; i++){
        Serial1.write(f2Map[i]/2);
    }
    Serial1.write(254);



  Serial1.write(255); Serial1.write(4);
  for(int i=0; i<8; i++){
      Serial1.write(crankEnrich[i]/2);
  }
  for(int i=0; i<8; i++){
      Serial1.write(afterStart[i]/2);
  }
  for(int i=0; i<8; i++){
      Serial1.write(warmUp[i]/2);
  }
  for(int i=0; i<8; i++){
      Serial1.write(MATenrich[i]);
  }
  for(int i=0; i<8; i++){
      Serial1.write(tpsdot[i]);
  }
  for(int i=0; i<8; i++){
      Serial1.write(rpmtps[i]);
  }
  for(int i=0; i<8; i++){
      Serial1.write(CLNTtps[i]/2);
  }
  Serial1.write(decay/3);
  Serial1.write(TPStresh/40);
  Serial1.write(rpmCoastLow/40);
  Serial1.write(rpmCoastHigh/40);
  Serial1.write(kPaCoast);
  Serial1.write(CoastCutDelay/100);
  for(int i=0; i<8; i++){
      Serial1.write(fuelTrimMult[i]);
  }
  for(int i=0; i<8; i++){
      Serial1.write(fuelTrimAdd[i]+100);
  }
  for(int i=0; i<8; i++){
      Serial1.write(injstart[i]/3);
  }
  Serial1.write(crankRPM/3);
  Serial1.write(afterStartTime);

  Serial1.write(revLimit/50);
```

```
      Serial1.write(softRevLimit/50);
      Serial1.write(revLimitTiming+50);
      Serial1.write(LaunchHardCut/50);
      Serial1.write(LaunchSoftCut/50);
      Serial1.write(launchTiming+50);
      Serial1.write(LaunchTPS);
      for(int i=0; i<8; i++){
          Serial1.write(idleDC[i]);
      }
      Serial1.write(fiCorr+50);


      Serial1.write(254);


      MapSync=0;
}




}



////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////




void EEPROMsync(){

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                                //EEPROM write
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////

byte EEPROMbuff[400];
int b=0;
static int address;
static unsigned int lastWrite;

if ( EEPROMWrite==1 ){


  if ( (millis()-lastWrite)>10 ){


  if ( address<352 ){                          ////////fuel map 1

    b=0;
    for(int i=0; i<18; i++){
         for(int j=0; j<18; j++){
           EEPROMbuff[b]=f1[i][j];
           b=b+1;
         }
       }

    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(6); //write enable
    digitalWrite(SPI2_NSS_PIN,HIGH);
```

```
    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(2); //write instruction
    SPI_2.transfer(address>>8);
    SPI_2.transfer(address);

    for (int i=0; i<32; i++){
       SPI_2.transfer(EEPROMbuff[i+address]);
    }

    digitalWrite(SPI2_NSS_PIN,HIGH);

    address=address+32;

  }
  else if ( address<704 ){                        /////////fuel map 2

    b=0;
    for(int i=0; i<18; i++){
         for(int j=0; j<18; j++){
           EEPROMbuff[b]=f2[i][j];
           b=b+1;
         }
       }

    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(6); //write enable
    digitalWrite(SPI2_NSS_PIN,HIGH);
    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(2); //write instruction
    SPI_2.transfer(address>>8);
    SPI_2.transfer(address);

    for (int i=0; i<32; i++){
       SPI_2.transfer(EEPROMbuff[i+address-352]);
    }

    digitalWrite(SPI2_NSS_PIN,HIGH);

    address=address+32;

  }
  else if ( address<1056 ){                       /////////ign map

    b=0;
    for(int i=0; i<18; i++){
         for(int j=0; j<18; j++){
           EEPROMbuff[b]=d[i][j];
           b=b+1;
         }
       }

    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(6); //write enable
    digitalWrite(SPI2_NSS_PIN,HIGH);
    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(2); //write instruction
    SPI_2.transfer(address>>8);
    SPI_2.transfer(address);

    for (int i=0; i<32; i++){
       SPI_2.transfer(EEPROMbuff[i+address-704]);
    }

    digitalWrite(SPI2_NSS_PIN,HIGH);

    address=address+32;

  }
  else if ( address<1088 ){

    digitalWrite(SPI2_NSS_PIN,LOW);
```

```
    SPI_2.transfer(6); //write enable
    digitalWrite(SPI2_NSS_PIN,HIGH);
    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(2); //write instruction
    SPI_2.transfer(address>>8);
    SPI_2.transfer(address);

    for (int i=0; i<18; i++){
       SPI_2.transfer(dRpm[i]/50);
    }

    digitalWrite(SPI2_NSS_PIN,HIGH);

    address=address+32;

  }
  else if ( address<1120 ){

    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(6); //write enable
    digitalWrite(SPI2_NSS_PIN,HIGH);
    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(2); //write instruction
    SPI_2.transfer(address>>8);
    SPI_2.transfer(address);

    for (int i=0; i<18; i++){
       SPI_2.transfer(dMap[i]/2);
    }

    digitalWrite(SPI2_NSS_PIN,HIGH);

    address=address+32;

  }
  else if ( address<1152 ){                    /////crank,afterstart,warmup enrichment

    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(6); //write enable
    digitalWrite(SPI2_NSS_PIN,HIGH);
    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(2); //write instruction
    SPI_2.transfer(address>>8);
    SPI_2.transfer(address);

    for (int i=0; i<8; i++){
       SPI_2.transfer(crankEnrich[i]/2);
    }
    for (int i=0; i<8; i++){
       SPI_2.transfer(afterStart[i]/2);
    }
    for (int i=0; i<8; i++){
       SPI_2.transfer(warmUp[i]/2);
    }

    digitalWrite(SPI2_NSS_PIN,HIGH);

    address=address+32;

  }
  else if ( address<1184 ){                    /////MAT,tpsdot,rpmtps

    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(6); //write enable
    digitalWrite(SPI2_NSS_PIN,HIGH);
    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(2); //write instruction
    SPI_2.transfer(address>>8);
    SPI_2.transfer(address);

    for (int i=0; i<8; i++){
```

```
      SPI_2.transfer(MATenrich[i]);
    }
    for (int i=0; i<8; i++){
       SPI_2.transfer(tpsdot[i]);
    }
    for (int i=0; i<8; i++){
       SPI_2.transfer(rpmtps[i]);
    }

    digitalWrite(SPI2_NSS_PIN,HIGH);

    address=address+32;

  }
  else if ( address<1216 ){                    /////CLNTtps, coast cut

    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(6); //write enable
    digitalWrite(SPI2_NSS_PIN,HIGH);
    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(2); //write instruction
    SPI_2.transfer(address>>8);
    SPI_2.transfer(address);

    for (int i=0; i<8; i++){
       SPI_2.transfer(CLNTtps[i]/2);
    }
    SPI_2.transfer(decay/3);
    SPI_2.transfer(TPStresh/40);
    SPI_2.transfer(rpmCoastLow/40);
    SPI_2.transfer(rpmCoastHigh/40);
    SPI_2.transfer(kPaCoast);
    SPI_2.transfer(CoastCutDelay/100);
    for(int i=0; i<8; i++){
         SPI_2.transfer(fuelTrimMult[i]);
     }

    digitalWrite(SPI2_NSS_PIN,HIGH);

    address=address+32;

  }
  else if ( address<1248 ){                    /////revlimit,launch

    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(6); //write enable
    digitalWrite(SPI2_NSS_PIN,HIGH);
    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(2); //write instruction
    SPI_2.transfer(address>>8);
    SPI_2.transfer(address);

    for(int i=0; i<8; i++){
         SPI_2.transfer(fuelTrimAdd[i]+100);
     }
    for(int i=0; i<8; i++){
         SPI_2.transfer(injstart[i]/3);
     }
    SPI_2.transfer(crankRPM/3);
    SPI_2.transfer(afterStartTime);
    SPI_2.transfer(revLimit/50);
    SPI_2.transfer(softRevLimit/50);
    SPI_2.transfer(revLimitTiming+50);
    SPI_2.transfer(LaunchHardCut/50);
    SPI_2.transfer(LaunchSoftCut/50);
    SPI_2.transfer(launchTiming+50);
    SPI_2.transfer(LaunchTPS);

    digitalWrite(SPI2_NSS_PIN,HIGH);

    address=address+32;
```

```
  }
  else if ( address<1280 ){                       /////idleDC,ficorr

    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(6); //write enable
    digitalWrite(SPI2_NSS_PIN,HIGH);
    digitalWrite(SPI2_NSS_PIN,LOW);
    SPI_2.transfer(2); //write instruction
    SPI_2.transfer(address>>8);
    SPI_2.transfer(address);

    for(int i=0; i<8; i++){
         SPI_2.transfer(idleDC[i]);
     }
    SPI_2.transfer(fiCorr+50);

    digitalWrite(SPI2_NSS_PIN,HIGH);

    address=address+32;
    address=0;
    EEPROMWrite=0;
  }




  lastWrite=millis();

  }

}



////////////////////////////////////////////////////////////////////////////////////////////////
/////////////
                                                //EEPROM read
////////////////////////////////////////////////////////////////////////////////////////////////
/////////////

if ( EEPROMRead==1 ){

  digitalWrite(SPI2_NSS_PIN,LOW);    ///////////////////////
  SPI_2.transfer(3);
  address=0;
  SPI_2.transfer(address>>8);
  SPI_2.transfer(address);

  for(int i=0; i<18; i++){
      for(int j=0; j<18; j++){
         f1[i][j]=SPI_2.transfer(15);
      }
   }
  digitalWrite(SPI2_NSS_PIN,HIGH);


  digitalWrite(SPI2_NSS_PIN,LOW);    ///////////////////////
  SPI_2.transfer(3);
  address=352;
  SPI_2.transfer(address>>8);
  SPI_2.transfer(address);

  for(int i=0; i<18; i++){
      for(int j=0; j<18; j++){
         f2[i][j]=SPI_2.transfer(15);
      }
   }
  digitalWrite(SPI2_NSS_PIN,HIGH);
```

```
digitalWrite(SPI2_NSS_PIN,LOW);    //////////////////////
SPI_2.transfer(3);
address=704;
SPI_2.transfer(address>>8);
SPI_2.transfer(address);

for(int i=0; i<18; i++){
    for(int j=0; j<18; j++){
       d[i][j]=SPI_2.transfer(15);
    }
 }
digitalWrite(SPI2_NSS_PIN,HIGH);


digitalWrite(SPI2_NSS_PIN,LOW);    //////////////////////
SPI_2.transfer(3);
address=1120;
SPI_2.transfer(address>>8);
SPI_2.transfer(address);

for (int i=0; i<8; i++){
   crankEnrich[i]=SPI_2.transfer(15)*2;
}
for (int i=0; i<8; i++){
   afterStart[i]=SPI_2.transfer(15)*2;
}
for (int i=0; i<8; i++){
   warmUp[i]=SPI_2.transfer(15)*2;
}
digitalWrite(SPI2_NSS_PIN,HIGH);


digitalWrite(SPI2_NSS_PIN,LOW);    //////////////////////
SPI_2.transfer(3);
address=1152;
SPI_2.transfer(address>>8);
SPI_2.transfer(address);

for (int i=0; i<8; i++){
   MATenrich[i]=SPI_2.transfer(15);
}
for (int i=0; i<8; i++){
   tpsdot[i]=SPI_2.transfer(15);
}
for (int i=0; i<8; i++){
   rpmtps[i]=SPI_2.transfer(15);
}
digitalWrite(SPI2_NSS_PIN,HIGH);


digitalWrite(SPI2_NSS_PIN,LOW);    //////////////////////
SPI_2.transfer(3);
address=1184;
SPI_2.transfer(address>>8);
SPI_2.transfer(address);

for (int i=0; i<8; i++){
   CLNTtps[i]=SPI_2.transfer(15)*2;
}
decay=SPI_2.transfer(15)*3;
TPStresh=SPI_2.transfer(15)*40;
rpmCoastLow=SPI_2.transfer(15)*40;
rpmCoastHigh=SPI_2.transfer(15)*40;
kPaCoast=SPI_2.transfer(15);
CoastCutDelay=SPI_2.transfer(15)*100;
for(int i=0; i<8; i++){
   fuelTrimMult[i]=SPI_2.transfer(15);
}
digitalWrite(SPI2_NSS_PIN,HIGH);
```

```
  digitalWrite(SPI2_NSS_PIN,LOW);    //////////////////////
  SPI_2.transfer(3);
  address=1216;
  SPI_2.transfer(address>>8);
  SPI_2.transfer(address);

  for(int i=0; i<8; i++){
       fuelTrimAdd[i]=SPI_2.transfer(15)-100;
   }
  for(int i=0; i<8; i++){
       injstart[i]=SPI_2.transfer(15)*3;
   }
  crankRPM=SPI_2.transfer(15)*3;
  afterStartTime=SPI_2.transfer(15);
  revLimit=SPI_2.transfer(15)*50;
  softRevLimit=SPI_2.transfer(15)*50;
  revLimitTiming=SPI_2.transfer(15)-50;
  LaunchHardCut=SPI_2.transfer(15)*50;
  LaunchSoftCut=SPI_2.transfer(15)*50;
  launchTiming=SPI_2.transfer(15)-50;
  LaunchTPS=SPI_2.transfer(15);
  digitalWrite(SPI2_NSS_PIN,HIGH);


  digitalWrite(SPI2_NSS_PIN,LOW);    //////////////////////
  SPI_2.transfer(3);
  address=1248;
  SPI_2.transfer(address>>8);
  SPI_2.transfer(address);

  for(int i=0; i<8; i++){
     idleDC[i]=SPI_2.transfer(15);
  }
  fiCorr=SPI_2.transfer(15)-50;
  digitalWrite(SPI2_NSS_PIN,HIGH);



  EEPROMRead=0;

}




}



////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////




////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
                                                    //VE calculation
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////


void VEcalc(){
```

```
if ( fuelMap==1 ){   ////////////////////////////Fuel map 1

for ( i=0 ; i<18 ; i++ ){
    float y1,y2;
      if ( kPa<f1Map[i] ){
        if ( kPa<f1Map[0] ){ Y=0; }
        else{
          y1=kPa-f1Map[i-1];
          y2=f1Map[i]-f1Map[i-1];
          Y=y1/y2+i-1;
          }
        break;
      }
  }
  if ( kPa>=f1Map[17] ){ Y=16.99; }

for ( i=0 ; i<18 ; i++ ){
   float x1,x2;
    if ( RPM<f1Rpm[i] ){
      if ( RPM<f1Rpm[0] ){ X=0; }
      else{
        x1=RPM-f1Rpm[i-1];
        x2=f1Rpm[i]-f1Rpm[i-1];
        X=x1/x2+i-1;
        }
      break;
    }
  }
  if ( RPM>=f1Rpm[17] ){ X=16.99; }

}

else{             ////////////////////////////Fuel map 2

for ( i=0 ; i<18 ; i++ ){
    float y1,y2;
      if ( kPa<f2Map[i] ){
        if ( kPa<f2Map[0] ){ Y=0; }
        else{
          y1=kPa-f2Map[i-1];
          y2=f2Map[i]-f2Map[i-1];
          Y=y1/y2+i-1;
          }
        break;
      }
  }
  if ( kPa>=f2Map[17] ){ Y=16.99; }

for ( i=0 ; i<18 ; i++ ){
   float x1,x2;
    if ( RPM<f2Rpm[i] ){
      if ( RPM<f2Rpm[0] ){ X=0; }
      else{
        x1=RPM-f2Rpm[i-1];
        x2=f2Rpm[i]-f2Rpm[i-1];
        X=x1/x2+i-1;
        }
      break;
    }
  }
  if ( RPM>=f2Rpm[17] ){ X=16.99; }

}
//////////////////////////////////////

  fX=X*10; fY=Y*10;

  Y1=Y;
  Y2=Y+1;
  X1=X;
```

```
  X2=X+1;

if ( fuelMap==1 ){
  Z1=f1[17-Y1][X1];
  Z2=f1[17-Y1][X2];
  Z3=f1[17-Y2][X1];
  Z4=f1[17-Y2][X2];
}
else{
  Z1=f2[17-Y1][X1];
  Z2=f2[17-Y1][X2];
  Z3=f2[17-Y2][X1];
  Z4=f2[17-Y2][X2];
}


  ZI=round(Z1*(X2-X)+Z2*(X-X1));
  ZII=round(Z3*(X2-X)+Z4*(X-X1));
  VE= ZI*(Y2-Y)+ZII*(Y-Y1);


}



////////////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                                    //Ignition timing calculation
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////

void IgnCalc(){


for ( i=0 ; i<18 ; i++ ){
    float y1,y2;
      if ( kPa<dMap[i] ){
        if ( kPa<dMap[0] ){ Y=0; }
        else{
          y1=kPa-dMap[i-1];
          y2=dMap[i]-dMap[i-1];
          Y=y1/y2+i-1;
          }
        break;
      }
  }
  if ( kPa>=dMap[17] ){ Y=16.99; }

for ( i=0 ; i<18 ; i++ ){
   float x1,x2;
    if ( RPM<dRpm[i] ){
      if ( RPM<dRpm[0] ){ X=0; }
      else{
        x1=RPM-dRpm[i-1];
        x2=dRpm[i]-dRpm[i-1];
        X=x1/x2+i-1;
        }
      break;
    }
  }
  if ( RPM>=dRpm[17] ){ X=16.99; }

dX=X*10; dY=Y*10;

Y1=Y;
Y2=Y+1;
X1=X;
X2=X+1;

Z1=d[17-Y1][X1];
Z2=d[17-Y1][X2];
```

```
Z3=d[17-Y2][X1];
Z4=d[17-Y2][X2];

ZI=round(Z1*(X2-X)+Z2*(X-X1));
ZII=round(Z3*(X2-X)+Z4*(X-X1));
fi= ZI*(Y2-Y)+ZII*(Y-Y1);
if ( fi > 60 ) { fi=60; }



}




////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////



void  coilCharge(void){

   if ( limit==0 ){
    digitalWrite(IGNpin,HIGH);
    //inj1O=chargeIndex+1;
    //injO=myT+timerT.getCount();
    }
   nextCoilCharge=injectorOrder[chargeIndex];
   timerI.detachInterrupt(1);
   coilChargeSch=0;

}



void coilFire(void) {

    digitalWrite(IGNpin,LOW);
    nextCoilFire=injectorOrder[fireIndex];
    //inj1C=fireIndex+1;
    //injC=myT+timerT.getCount();

    ignEvent=1;
    timerI.detachInterrupt(2);
    coilFireSch=0;
    CAMfilter=micros();
}




////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////



void injOpen(void){

 uint16_t timerCCR;
```

```
   if ( injector==1 ){
    digitalWrite(injPin1,HIGH);
    timerCCR = TIMER4->regs.gen->CNT + Tinj1;
    TIMER4->regs.gen->CCR1 = timerCCR;                  //inj1O=myT+timerT.getCount();
    timerIV.attachInterrupt(1,inj1close);
    TIMER4->regs.gen->SR &= ~0x0002;
    nextInjOpen=injectorOrder[0];
    }
   else if ( injector==2 ){
    digitalWrite(injPin2,HIGH);
    timerCCR = TIMER4->regs.gen->CNT + Tinj2;
    TIMER4->regs.gen->CCR2 = timerCCR;                  //inj2O=myT+timerT.getCount();
    timerIV.attachInterrupt(2,inj2close);
    TIMER4->regs.gen->SR &= ~0x0004;
    nextInjOpen=injectorOrder[1];
    }
   else if ( injector==3 ){
    digitalWrite(injPin3,HIGH);
    timerCCR = TIMER4->regs.gen->CNT + Tinj3;
    TIMER4->regs.gen->CCR3 = timerCCR;                  //inj3O=myT+timerT.getCount();
    timerIV.attachInterrupt(3,inj3close);
    TIMER4->regs.gen->SR &= ~0x0008;
    nextInjOpen=injectorOrder[2];
    }
   else if ( injector==4 ){
    digitalWrite(injPin4,HIGH);
    timerCCR = TIMER4->regs.gen->CNT + Tinj4;
    TIMER4->regs.gen->CCR4 = timerCCR;                  //inj4O=myT+timerT.getCount();
    timerIV.attachInterrupt(4,inj4close);
    TIMER4->regs.gen->SR &= ~0x0010;
    nextInjOpen=injectorOrder[3];
    }
   else if ( injector==5 ){
    digitalWrite(injPin5,HIGH);
    timerCCR = TIMER2->regs.gen->CNT + Tinj5;
    TIMER2->regs.gen->CCR1 = timerCCR;                  //inj5O=myT+timerT.getCount();
    timerII.attachInterrupt(1,inj5close);
    TIMER2->regs.gen->SR &= ~0x0002;
    nextInjOpen=injectorOrder[4];
    }
   else if ( injector==6 ){
    digitalWrite(injPin6,HIGH);
    timerCCR = TIMER2->regs.gen->CNT + Tinj6;
    TIMER2->regs.gen->CCR2 = timerCCR;                  //inj6O=myT+timerT.getCount();
    timerII.attachInterrupt(2,inj6close);
    TIMER2->regs.gen->SR &= ~0x0004;
    nextInjOpen=injectorOrder[5];
    }

    injector=0;
    timerII.setCompare( 3, timerII.getCount()+65000 );

}

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////

void inj1close(void){
   digitalWrite(injPin1,LOW);  //inj1C=myT+timerT.getCount();
   timerIV.detachInterrupt(1);
}

void inj2close(void){
   digitalWrite(injPin2,LOW);  //inj2C=myT+timerT.getCount();
   timerIV.detachInterrupt(2);
}

void inj3close(void){
```

```
   digitalWrite(injPin3,LOW);  //inj3C=myT+timerT.getCount();
   timerIV.detachInterrupt(3);
}

void inj4close(void){
   digitalWrite(injPin4,LOW);  //inj4C=myT+timerT.getCount();
   timerIV.detachInterrupt(4);
}

void inj5close(void){
   digitalWrite(injPin5,LOW);  //inj5C=myT+timerT.getCount();
   timerII.detachInterrupt(1);
}

void inj6close(void){
   digitalWrite(injPin6,LOW);  //inj6C=myT+timerT.getCount();
   timerII.detachInterrupt(2);
}




////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////




void IGN() {

static float deltaSplit,lastRPMsplit;
uint16_t timerCCR;


   if ( btrigger<10 ) { btrigger=btrigger+1; }
   RPMsplit=myT + TIMER3->regs.gen->CNT - RPMtime;
   deltaSplit=lastRPMsplit/RPMsplit;
   if ( deltaSplit<2 || btrigger<4 ){
        //GMTtime=RPMsplit;
        //CAMsignal=digitalRead(CAMpin);
   RPMtime=myT + TIMER3->regs.gen->CNT;
if ( btrigger>3 )  {
   triggersplit=RPMsplit-1.8*prevRPMsplit;
   prevRPMsplit=RPMsplit;
   if ( Synced==0 ){
    if ( triggersplit > 0 && digitalRead(CAMpin)==1 ){ crankpos=0; Synced=1;
    RPMsplit=RPMsplit/(1+missingTooth); }
    else if ( triggersplit > 0 && digitalRead(CAMpin)==0 ) { crankpos=teethNum; Synced=1;
    RPMsplit=RPMsplit/(1+missingTooth); }
    if (  CAMsync==1 ){ crankpos=0; CAMsync=0; Synced=1; }
   }
   else if ( triggersplit > 0 /*|| CAMsync==1*/ ) {
    crankpos=crankpos+missingTooth; RPMsplit=RPMsplit/(1+missingTooth);
    if ( crankpos>teethNumX2 ) { error=error+1; }
    if ( crankpos>teethNumX2_1 /*&& digitalRead(CAMpin)==1*/ ){ crankpos=0; CAMsync=0;
    ciclusTime=micros(); }
   }

   crankpos=crankpos+1;
   CAMsync=0;


   //if ( crankpos==1 ){ vrijeme2=myT+TIMER3->regs.gen->CNT; }

   if ( crankpos==(teethNum-missingTooth) || crankpos==(teethNumX2-missingTooth) ){
```

```
    timerCCR=TIMER2->regs.gen->CNT + RPMsplit-4;
    TIMER2->regs.gen->CCR4 = timerCCR;
    if ( missingTooth>0 ){ timerII.attachInterrupt(4,virtualTrigger); }
    TIMER2->regs.gen->SR &= ~0x0010;
    }


  if ( crankpos==prevchargePos && coilChargeSch==0 ) {
        timerCCR=TIMER1->regs.gen->CNT + prevTcharge-5;
        TIMER1->regs.gen->CCR1 = timerCCR; noInterrupts();
        timerI.attachInterrupt(1, coilCharge);
        TIMER1->regs.gen->SR &= ~0x0002; interrupts();
        coilChargeSch=1;
        }

  if ( crankpos==prevfirePos && coilFireSch==0 ) {
        timerCCR=TIMER1->regs.gen->CNT + prevTfire-5;
        TIMER1->regs.gen->CCR2 = timerCCR; noInterrupts();
        timerI.attachInterrupt(2, coilFire);
        TIMER1->regs.gen->SR &= ~0x0004; interrupts();
        coilFireSch=1;
        }


}

}
else { RPMsplit=lastRPMsplit; }

lastRPMsplit=RPMsplit;

}




void CAMtrigg(){

if ( btrigger>3 && (micros()-CAMfilter)>500 ){
  //CAMsync=1;
  //vrijeme2=myT+TIMER3->regs.gen->CNT;
}

}




////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////




void virtualTrigger (void) {


  static int virtCrankpos,virtTrigger;
  uint16_t timerCCR;

  timerCCR=TIMER2->regs.gen->CNT + RPMsplit;
  TIMER2->regs.gen->CCR4 = timerCCR;

  virtTrigger=virtTrigger+1;
  virtCrankpos=crankpos+virtTrigger;
```

```
  if ( virtTrigger==missingTooth ){ timerII.detachInterrupt(4); virtTrigger=0; }

  if ( virtCrankpos==prevchargePos && coilChargeSch==0 ) {
        timerCCR=TIMER1->regs.gen->CNT + prevTcharge-3;
        TIMER1->regs.gen->CCR1 = timerCCR; noInterrupts();
        timerI.attachInterrupt(1, coilCharge);
        TIMER1->regs.gen->SR &= ~0x0002; interrupts();
        coilChargeSch=1;
        }

  if ( virtCrankpos==prevfirePos && coilFireSch==0 ) {
        timerCCR=TIMER1->regs.gen->CNT + prevTfire-4;
        TIMER1->regs.gen->CCR2 = timerCCR; noInterrupts();
        timerI.attachInterrupt(2, coilFire);
        TIMER1->regs.gen->SR &= ~0x0004; interrupts();
        coilFireSch=1;
        }



}




////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////




////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
                                                    //VTEC
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////

void VTECcalc(){

  int tpsVTEC;
  static unsigned int VTECtime,VTECsplit;

  tpsVTEC=-0.0166*RPM+75;
  if ( tpsVTEC<0 ){ tpsVTEC=0; }

  if ( RPM>2500 && TPSperc>tpsVTEC ){ digitalWrite(VTECpin,LOW); VTECtime=millis();
  fuelMap=2; }
     else if ( RPM<2300 /*|| TPSperc<4*/ || VTECsplit>4000 ) { digitalWrite(VTECpin,HIGH);
     fuelMap=1; }

  VTECsplit=millis()-VTECtime;

  //if ( TPSperc>50 ){ digitalWrite(VTECpin,LOW); }

}


////////////////////////////////////////////////////////////////////////////////////////////////////
///////////
                                                    //Water injection
////////////////////////////////////////////////////////////////////////////////////////////////////
///////////


void WIcalc(){
```

```
if ( kPa > kpaWI ){ digitalWrite(WIpin,HIGH); }
   else if ( kPa < (kpaWI-5) ){ digitalWrite(WIpin,LOW); }

if ( WItest==1 ){ digitalWrite(WIpin,HIGH); }


}


////////////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                                    //IAC
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////

void IACcalc(){
  static int IACintEnabled;


  Ytemp=0.06667*CLNTVal+1.33;
  if ( Ytemp<0.01 ){ Ytemp=0.01; }
  if ( Ytemp>6.99 ){ Ytemp=6.99; }
  Ytemp1=Ytemp; Ytemp2=Ytemp1+1;
  IACdc=idleDC[Ytemp1]*(Ytemp2-Ytemp)+idleDC[Ytemp2]*(Ytemp-Ytemp1);

  Input=avgRPM;
  Setpoint=940;
  device=2;
  //if ( avgRPM>500 && avgRPM<1300 && TPSperc<1 ){ IACdc=calcPID(iacKp,iacKi)+IACdc; }
  if ( antiLag==1 && TPSperc<5 ){ IACdc=75; }
  if ( RPM>400 && RPM<1800 && engineON<1 ){ IACdc=85; }
  if ( IACdc>98 ){ IACdc=98; }

  IACdcHigh=IACdc*0.01*IACperiod;
  IACdcLow=IACperiod-IACdcHigh;

  if ( IACdc==0 && IACintEnabled==1 ){
    timerV.detachInterrupt(1);
    digitalWrite(IACpin,LOW);
    IACintEnabled=0;
    }
  else if ( IACdc>0 && IACintEnabled==0 ){
    timerV.attachInterrupt(1, IACpwm);
    IACintEnabled=1;
    }

}



void IACpwm(){

  uint16_t timerCCR;

  if ( IACstate==0 ){
    digitalWrite(IACpin,HIGH);
    timerCCR=TIMER5->regs.gen->CNT+IACdcHigh;
    TIMER5->regs.gen->CCR1 = timerCCR;
    IACstate=1;
  }
  else{
    digitalWrite(IACpin,LOW);
    timerCCR=TIMER5->regs.gen->CNT+IACdcLow;
    TIMER5->regs.gen->CCR1 = timerCCR;
    IACstate=0;
  }

}

////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////
                                                    //EBC
//////////////////////////////////////////////////////////////////////////////////////////////
////////////

void EBCcalc(){

static byte EBC;
static unsigned int EBCtime;

EBCdc=0;

if ( TPSperc>80 ) {
  Input=kPa;
  Setpoint=180;
  device=1;
  EBCdc=calcPID(ebcKp,ebcKi);
 }


 if ( (millis()-EBCtime)>65 ) {
     if ( TPSperc>80 ) { digitalWrite(EBCpin,HIGH); }
     EBC=1;
     EBCtime=millis();
  }

 if ( (millis()-EBCtime)>EBCdc && EBC==1 ) {
     digitalWrite(EBCpin,LOW);
     EBC=0;
  }


}



//////////////////////////////////////////////////////////////////////////////////////////////
////////////
                                                    //PID
//////////////////////////////////////////////////////////////////////////////////////////////
////////////


int calcPID(float Kp, float Ki) {

  int Output,error,checkOut;
  static int errorSumIAC,errorSumEBC,prevEBCout,prevIACout;
  static unsigned int ebcTime,iacTime;

  error=Setpoint-Input;

  if ( device==1 ){ Output=prevEBCout; }
  if ( device==2 ){ Output=prevIACout; }


  if ( device==1 && (micros()-ebcTime)>5000 ){   //EBC
    if ( Ireset==1 ){ errorSumEBC=0; Ireset=0; }
    checkOut=Kp*error + errorSumEBC;
    if ( checkOut>0 && checkOut<67 ){ errorSumEBC=errorSumEBC + error*Ki; }
    Output=Kp*error + errorSumEBC;
    if ( Output<0 ){ Output=0; }
    if ( Output>67 ){ Output=67; }
    ebcTime=micros();
    prevEBCout=Output;
  }

  if ( device==2 && (micros()-iacTime)>5000 ){  //IAC
    if ( Ireset==1 ){ errorSumIAC=0; Ireset=0; }
    checkOut=Kp*error + errorSumIAC;
    if ( checkOut>0 && checkOut<100 ){ errorSumIAC=errorSumIAC + error*Ki; }
    Output=Kp*error + errorSumIAC;
```

```
    if ( Output<0 ){ Output=0; }
    if ( Output>98 ){ Output=98; }
    iacTime=micros();
    prevIACout=Output;
  }


  return Output;

}
```